SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Matija Polgar

Zagreb, 2012.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Bojan Jerbić, dipl. ing.

Student:

Matija Polgar

Zagreb, 2012.

Izjavljujem da sam ovaj rad izradio samostalno, koristeći stečena znanja tijekom studija i navedenu literaturu.

Zahvaljujem mentoru prof. dr. sc. Bojanu Jerbiću na savjetima i pomoći tijekom izrade diplomskoga rada. Također zahvaljujem i asistentu Tomislavu Stipančiću na velikoj pomoći prilikom izrade zadataka diplomskoga rada.

Zahvaljujem teti Anđeli i tetku Stjepanu Raiču koji su mi pružili veliku pomoć, kako financijsku tako i moralnu tijekom studija.

Posebnu zahvalnost dugujem svojim roditeljima, majci Snježani i ocu Zdenku na velikom strpljenju, odricanju i potpori koju su mi pružali tijekom moga školovanja i studiranja.

Matija Polgar

SADRŽAJ

POPIS SLIKA

SAŽETAK

U ovom diplomskom radu riješen je problem prepoznavanja više objekata i povezivanje programskoga rješenja s dostupnom IP kamerom smještenom u Laboratoriju za projektiranje izradbenih i montažnih sustava na Fakultetu strojarstva i brodogradnje u Zagrebu. U svrhu izrade programskog rješenja korišten je C++ programski jezik s OpenCV bibliotekom otvorenog koda.

Prvi dio rada je kratki osvrt na računalni vid i probleme računalnoga, prije svega prepoznavanje objekata. Spomenuti su neki od algoritama za traženje ključnih točaka i njihovo povezivanje, upoznavanje s mogućnostima i razvojem OpenCV biblioteke i na posljetku je riječ o IP kameri i njezinim prednostima i nedostatcima.

Drugi se dio rada odnosi na izradu zadatka. Program za prepoznavanje objekata u suštini se može svesti na tri glavna algoritma, a to su SURF algoritam za pronalaženje ključnih točaka i njihovo opisivanje deskriptorima, k-NN algoritam za pronalaženje mogućih parova ključnih točaka na različitim slikama i na kraju homografija za prostorno povezivanje točaka. U radu su detaljno objašnjeni algoritmi i njihova primjena na program prepoznavanja objekata. Nakon izrade programa u radu je prikazano i rješenje spajanja programa s IP kamerom.

Na kraju rada prikazani su rezultati pretrage scena.

1. UVOD

U što većem nastojanju dovođenja računalne inteligencije ukorak s ljudskom, znanstvenici razvijaju razne tehnike i algoritme koji omogućuju računalu mogućnost raspoznavanja svijeta oko sebe. Većina se algoritama temelji na proučavanjima ljudskoga ponašanja i na određenim prirodnim pojavama. Nama, ljudima, racionalnim bićima, neki su procesi jednostavni i prirodni. No, kako računalo to može shvatiti?

Unutar računala, u njegovu procesoru, „mozgu“ računala, odvija se niz matematičkih radnji za obavljanje bilo kakvih zadataka. Neke od tih zadataka računalo obavlja nezamislivo brže nego čovjek, primjerice množenje brojeva, dok neke zadatke gdje je potrebno razmišljanj, računalo ne može obaviti. Ljudski je mozak složeni organ i najvažniji za ljudsku sposobnost razmišljanja. Možemo se pitati zašto u računalo ne ugradimo jedan takav? Odgovor je da ne možemo, jer se ljudski mozak evolucijski razvijao milijunima godina i neki procesi koji se u njemu odvijaju i dan-danas su nepoznati.

Ovaj se rad bavi problemima računalnoga vida i raspoznavanja objekata. Budući da smo mi vizualna bića, lako se možemo prevariti da su zadatci računalnoga vida jednostavni. Koliko nama može biti teško pronaći npr. auto gledajući u sliku? Početna intuicija može biti vrlo pogrešna. Ljudski mozak dijeli vidni signal na mnogo kanala koji prenose različite informacije u naš mozak. Mozak ima sustav koji utvrđuje bitne dijelove na slici dok ostale dijelove potiskuje. U mozgu postoji mnogo povratnih i informacija u vidnome toku što se još uvijek premalo razumije. Postoji i mnogo ulaznih podataka iz senzora kontroliranih mišićima i ostalih senzora koji omogućuju mozgu da ih poveže s iskustvima godinama življenja na svijetu. Petlje povratnih informacija u mozgu idu kroz sve faze obrade, uključujući i same senzore (oči) koje mehanički kontroliraju osvjetljenje kroz šarenicu oka i podražuju receptore na površini mrežnice.

U sustavu računalnoga vida računalo prima mrežu brojeva od kamere. Za mnoge dijelove ne postoji ugrađeni uzorak prepoznavanja, nema automatske kontrole fokusa i otvaranja optičke leće, nema povezivanja s iskustvima iz prijašnjih godina. Za mnoge dijelove računalni vid još nije dovoljno vješt. U toj mreži brojeva koje računalo vidi, odnosno u svakom broju postoji mnogo šumova i zbog toga računalo dobiva malo korisnih informacija, ali ta mreža je ipak

sve što računalo „vidi“. Upravo je zadatak pretvoriti tu mrežu brojeva punu šumova u nešto što će računalo shvatiti.

Korištenjem dosad postojećih algoritama mogu se riješiti mnogi problemi računalnoga vida. Rad se bavi konkretno problemom prepoznavanja objekata na scenama, pa će se u skladu s tim koristiti neki od algoritma koji se odnose na to područje. Prije svega SURF algoritam pomoću kojega će se tražiti ključne točke i konstruirati deskriptori, nakon toga tražit će se parovi deskriptora pomoću k-NN algoritma i na kraju, homografijom će se povezati kutne točke između slika

Za „vid“ računala koristit će se IP kamera preko koje će se pristupati sa servera. Kamera će pretraživati scene na kojima će se nalaziti objekti i pomoću programa za prepoznavanje dobivat će se rezultati pretrage koji će reći nalazi li se objekt ili ne nalazi na sceni.

2. RAČUNALNI VID

Kao ljudi, mi imamo trodimenzionalni pogled na svijet oko nas očiglednom lakoćom. Gledajući uokvirenu sliku, lako možemo prepoznati i prebrojati osobe na slici, pa čak i pogoditi njihova emocionalna stanja iz njihovih izraza lica. Stručnjaci su desetljećima pokušavali ustanoviti kako vizualni sustav radi, čak iako mogu osmisliti optičke iluzije kako bi opovrgavali neke principe, cijeli taj sustav ostaje i dalje zagonetan i nedosegnut.

Istraživači u područjima računalnoga vida paralelno razvijaju matematičke tehnike za dobivanje trodimenzionalnih oblika i pojavu objekata na slikama. Danas postoje pouzdane tehnike ta računanje djelomičnih 3D modela okoline od nekoliko stotina djelomičnih fotografija koje se preklapaju. Ako ima dovoljan set pogleda na objekt, može se točno izraditi 3D model objekta koristeći stereo-povezivanje. Mogu se pratiti kretnje osoba, pa čak se mogu i na slici prepoznati i imenovati osobe, kombinirajući njihova lica, odjeću i detekciju i prepoznavanje kose. No, unatoč svim tim napredcima imati računalo koje će interpretirati sliku kao, recimo, dvogodišnje dijete, ostaje nedostižno.

Zašto je uopće viđenje tako teško? Dijelom zato što je vid obrnuti problem u kojemu tražimo povrat nekih nepoznatih i nedovoljnih informacija kako bi potpuno odredili rješenje. Stoga je potrebno osloniti se na fizičke i probabilističke modele kako bi se mogla pronaći potencijalna rješenja. Danas se računalni vid koristi u mnogim područjima stvarnoga svijeta:

- Pregledavanje strojeva: brzi pregled dijelova za osiguranje kvalitete, koristeći stereo-vid sa specijaliziranim osvjetljenjem za mjerenje tolerancija na dijelovima ili traženje oštećenja pri lijevanju željeza koristeći x-zrake.
- Trgovina: prepoznavanje objekata za automatsku naplatu.
- Medicinske slike: pregledavanje preoperativnih i intraoperativnih slika ili obavljanje dugotrajnih studija morfologije ljudskoga mozga.
- Automobilska sigurnost: otkrivanje neočekivanih prepreka, poput pješaka na ulicama, u slučajevima kada tehnike poput radara ne rade kako treba.
- Povezivanje pokreta: spajanjem računalnogeneriranih slika za snimanje u stvarnom vremenu, prateći točke značajki izvornoga videa za procjenu kretnji i okoline 3D kamerama. Takve se tehnike koriste u Hollywoodu pri snimanju filmova.
- Nadzor: nadzor za uljeze, analiziranje prometa, nadzor bazena zbog žrtava utapanja.

Sve su to vrlo važne primjene, a uglavnom se odnose na specijalizirane vrste slika za uska područja primjene.

2.1. Kratka povijest računalnoga vida

Kad se računalni vid pojavio u ranima 1970-im godinama, na njega se gledalo kao na vizualnu komponentu ambicioznoga programa oponašanja ljudske inteligencija i kako će robot dobiti inteligentno ponašanje. U početku su mnogi vjerovali da je rješavanje „vizualnog ulaza“ lakši dio rješavanja problema, dok su rasuđivanje i planiranje više razine puno teži problemi.

Prvi su pokušaji uključivali tehnike izvlačenja rubova i potom određivanje 3D strukture objekta na temelju topologijske strukture 2D linija. Napravljeno je i nekoliko algoritama koji su dali dobar pogled na to novo područje

U 1980-ima se mnogo pažnje posvetilo naprednijim matematičkim tehnikama za obavljanje kvantitativnih analiza slika i scena. Piramide slika počele su se naširoko rabiti za obavljanje zadataka kao što su stapanje slika, traženje sličnosti na slikama. Daljnjom uporabom piramida slika razvio se i koncept prostora skala. U tom su periodu bila i istraživanja za bolje otkrivanje rubova i kontura. Istraživači otkrivaju da se mnogi algoritmi za otkrivanje značajki mogu povezati s istim matematičkim okvirom ako se postave kao optimizacijski problemi.

Idućih deset godina nastavlja se na istraživanjima i poboljšavanjima. Stereo-vid postaje sve značajnija tema istraživanja. Mnogo se radi na algoritmima za praćenje, posebice praćenje kontura.

Ulaskom u novo tisućljeće i dalje se nastavljaju istraživanja u svim granama računalnoga vida. Posebice se pridaje važnost tehnikama prepoznavanja objekata na temelju značajki. Te tehnike dominiraju također i u prepoznavanju prizora, pa čak i panorama. Veliko značenje se daje i prepoznavanju lokacija utemeljeno na konturama ili segmentima prostora.

2.2. Prepoznavanje i povezivanje značajki

Prepoznavanje i povezivanje značajki jedne su od najvažnijih komponenti mnogih primjena računalnoga vida. Ako želimo povezati dvije slike, prvo trebamo odrediti neke značajke na slici kako bi hi mogli povezivati.

Postoji nekoliko vrsta značajki. Prva su vrsta neke specifične lokacije na slikama. Te se lokalizirane značajke često nazivaju ključne značajke ili značajne točke i često su opisane pojavom nakupina piksela oko lokacije značajne točke. Druga vrsta bitnih značajki su rubovi. Ta vrsta značajki može biti povezivana na temelju svojstvenih orijentacija i lokalnih pojava i također može biti dobar pokazatelj za granice objekta. Rubovi se mogu svrstati u segmente zakrivljenja i pravocrtnih linija, koji se mogu izravno povezivati ili analizirati kao bi se našle točke a time i vanjski i unutarnji parametri kamere. I na kraju, velika područja jednakih boja ili tekstura mogu biti izvučena kao zasebne cjeline i mogu se koristiti za razna povezivanja i manipulaciju slike.

2.2.1. Točke

Značajne točke često se koriste za traženje razbacanih setova odgovarajućih lokacija na različitim slikama i često su pokazivači za računanje pozicija kamera, što je preduvjet za računanje gušćega skupa zajedničkih točaka kad se koristi stereo-povezivanje. Takva se povezivanja mogu rabiti za uspoređivanje slika. Također se koriste i za prepoznavanje pojave i vrste objekata. Glavna prednost ključnih točaka jest ta da i dalje omogućuju povezivanja, čak i kod većih promjena razina i orijentacija slika.

Postoje dva glavna pristupa u traženju značajnih točaka i njihovih povezivanja. Prvi je traženje značajki na jednoj slici koji se mogu točno pratiti koristeći lokalne tehnike pretraživanja kao što su korelacija ili najmanji kvadrat. Drugi pristup je neovisno otkrivanje značajki na svim slikama i njihovo povezivanje temeljeno na lokalnim pojavama.

Prepoznavanje objekata može se podijeliti u tri osnovne faze. Prva ja otkrivanje i izdvajanje točaka; svaka se slika pretražuje za lokacije koje će najvjerojatnije biti dobro povezane s drugim slikama. Druga je faza opisivanje točki; svaka je regija oko pronađenih točaka

pretvorena u deskriptor koji se kasnije može povezivati s drugim deskriptorima. Treća faza je povezivanje točaka, efektivno traženje najboljih kandidata za povezivanje sa drugim slikama.

2.2.1.1. Detektori značajki

Kod prepoznavanja značajki najčešće se postavlja pitanje koje su značajke najbolje za povezivanje? Ako pogledamo sliku 1., vidjet ćemo dvije slike s istim kvadratnim dijelovima slike koji se povezuju. Dijelove koji nemaju teksturu gotovo je nemoguće lokalizirati. Dijelove koji imaju velike promjene u kontrastu (gradijente), lakše je lokalizirati, iako za segmente s ravnim linijama pri jednoj orijentaciji dolazi do problema jer je moguće uskladiti samo dijelove uzduž smjera koji je u skladu sa smjerom ruba. Dijelove s najmanje dvije različite orijentacije najlakše je lokalizirati.

Slika 1. Par slika s izvučenim kvadratnim dijelovima (dolje)

Förstner-Harris

Förstner i Harris su prvi predložili korištenje lokalnih maksimuma u rotacijski invarijantinm skalarnim mjerama deriviranima iz auto-korelacijske matrice kako bi se locirale ključne točke. Ta tehnika koristi Gaussove težinske okvire umjesto navedenih kvadratnih dijelova slike, što čini odziv detektora neosjetljivim na rotacije slike.

Adaptive non-maximal susppression

Mnogi detektori značajki traže lokalni maksimum u značajnoj funkciji i to može dovesti do neparnih raspodjela točaka preko slike, npr. točke će biti gušće u regijama s većim kontrastom. Da bi se ublažio taj problem, detektiraju se značajke koje su lokalni maksimum i čiji je odziv značajno veći od odziva svih susjeda u području radijusa r. Za to postoji efikasan

način pridruživanja potiskivanih radijusa sa svim lokalnim maksimumima, sortirajući prvo sve lokalne maksimume po njihovoj snazi odziva, a onda kreirajući drugu listu sortiranu smanjenjem potiskivanoga radijusa.

2.2.1.2. Deskriptori značajki

Nakon otkrivanja značajki (ključnih točaka), slijedi povezivanje. Odrediti koje značajke dolaze iz odgovarajućih lokacija na različitim slikama. U mnogim će se slučajevima lokalna pojava značajki mijenjati u orijentaciji, skali, pa čak i Affinom okviru između slika. Međutim, prije formiranja deskriptora dobro je odrediti lokalnu skalu i orijentaciju. Postoji više deskriptora od kojih su najznačajniji: MOPS, SIFT, PCA-SIFT, GLOH i SURF.

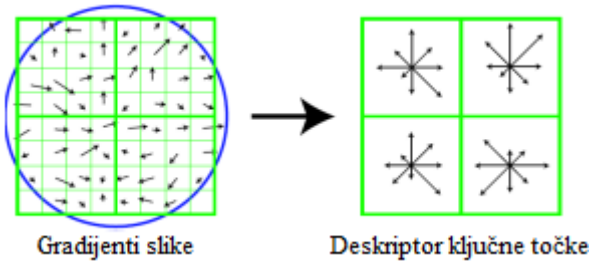
MOPS

Za jednostavnije zadatke koji ne trebaju velika skraćivanja, kao što je vezanje slika, najjednostavnije je rabiti jednostavne okvire. Kako bi se nadoknadile male netočnosti u detektoru značajke, ti višeskalni orijentirani okviri uzorkovani su na razmaku od 5 piksela relativno u odnosu na skalu pronalaženja. Za kompenziranje Affinih fotometrijskih varijacija intenziteti okvira se ponovno skaliraju pa je njihova oznaka nula, a njihov je otklon jedan.

SIFT

SIFT značajke se oblikuju računanjem gradijenta svakog pikslea u 16x16 okviru oko pronađene ključne točke, koristeći prikladnu razinu Gaussove piramide kod koje su ključne točke pronađene. Veličina gradijenata smanjuje težinu Gaussovom funkcijom kako bi se smanjio utjecaj gradijenata koji su dalje od centra, jer su oni pod većim utjecajem manjih pogrešnih otkrivanja.

U svakom 4x4 kvadratu formira se histogram orijentacije gradijenta dodajući vrijednost težine gradijenta jednom od 8 polja orijentacije histograma. Kako bi se smanjio utjecaj lokacija i dominantne pogrešne procjene orijentacije, svaki od 256 težinskih veličina gradijenata dodaje se u 2x2x2 polje histograma koristeći se prostornom interpolacijom. Rezultat je 128-D vektor deskriptora prikazan na slici 2.



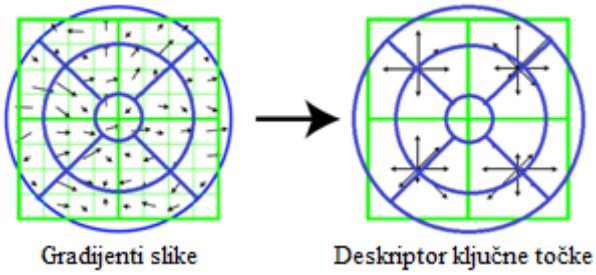
Slika 2. Shematski prikaz SIFT-a

PCA-SIFT

PCA-SIFT deskriptori su nadahnuti SIFT deskriptorom, a računaju se kroz x i y derivacije preko 39x39 okvira i potom smanjuju, rezultirajući 3042-dimenzionalni vektor na 36-D vektor koristeći PCA (*Principal Component Analysis*) metodu.

GLOH (Gradient Location-Orientattion Histogram)

GLOH deskriptor je varijanta SIFT deskriptora koja umjesto 4 kvadrata koristi polarna polja. Prostorna polja su radijusa 6, 11 i 15 s osam uglatih površina (osim središnje regije) i s ukupno 17 prostornih polja i 16 orijentacijskih polja (slika 3.). 272-dimenzionalni histogram daje 128-dimenzionalni deskriptor, koristeći PCA metodu na velikoj bazi podataka.



Slika 3. Shematski prikaz GLOH-a

U programu za prepoznavanje objekata koristit će se SURF detektor. SURF algoritam će biti detaljnije opisan u poglavlju 6.

2.2.1.3. Povezivanje značajki

Jednom kad su pronađene značajke i izračunati njihovi deskriptori na jednoj ili više slika, sljedeći korak je uspostaviti neka preliminarna povezivanja značajki između tih slika. Pristup tomu ovisi djelomično o primjeni, npr. različite strategije mogu biti poželjnije za različito povezivanje slika za koje se zna da se poklapaju ili povezivanje slika koje možda uopće nemaju sličnosti. Problem povezivanja se može podijeliti na dva dijela. Prvo je određivanje strategije povezivanja, koja određuje koje su te odgovarajuće značajke koje se povezuju i koje ulaze u sljedeći korak procesa. Drugi dio je smišljanje najbolje strukture podataka i algoritama za obavljanje povezivanja s najkraćim vremenom.

2.2.1.3.1. Strategija povezivanja

Ako pretpostavimo da imamo dvije slike koje se dosta dobro preklapaju, znamo da će se većina značajki na jednoj slici vjerojatno dobro povezivati sa značajkama na drugoj slici, iako se neke možda neće povezati jer su ili zatvorene ili je njihova pojava previše izmijenjena. S druge strane, ako želimo prepoznati koliko se poznatih objekata nalazi na sceni, većina značajki se možda neće uspjeti povezati. Veliki se broj potencijalnih objekata mora pretražiti, što zahtijeva učinkovitije strategije.

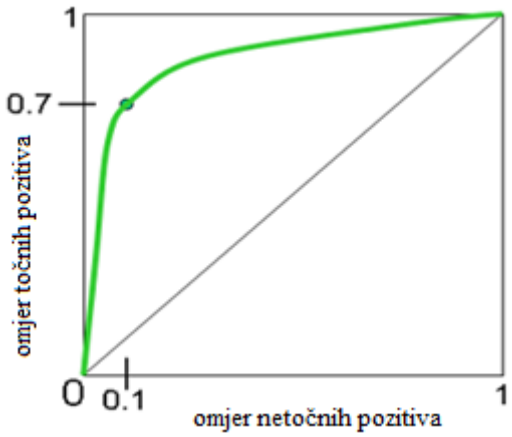
Prvo ćemo pretpostaviti da su određeni deskriptori i da se Euklidova udaljenost u prostoru značajke može uzeti za rangiranje povezivanja značajki. Zadanom Euklidovom udaljenošću, najjednostavnija strategija povezivanja je namjestiti *treshold* (maksimalnu granicu) i vratiti sve povezane značajke s druge slike unutar te granice. Prevelika granica rezultira većim brojem netočnih pozitiva (povezanih značajki), premala granica rezultira većim brojem netočnih negativa (nepovezanih značajki). Te brojeve možemo staviti u omjer definirajući točni omjer pozitiva:

$$omjer\ točnih\ pozitiva = \frac{\#točnih\ pozitiva}{\#povezanih\ značajki(pozitivi)} = 1 - \frac{\#netočnih\ negativa}{\#povezanih\ značajki(pozitivi)}$$

i netočni omjer pozitiva:

$$omejr\ netočnih\ pozitiva = \frac{\#netočni\ pozitivi}{\#nepovezane\ značajke(negativi)}$$

Sve pojedine strategije povezivanja mogu se ocijeniti s ta dva omjera. Idealno je ako je omjer točnih pozitiva bliže broju 1, a omjer netočnih pozitiva bliži 0.



Slika 4. ROC krivulja

Na slici je prikazana ROC (*Receiver Operating Characteristic*) krivulja koja iscrtava omjer točnih pozitiva naspram omjeru netočnih pozitiva za određenu kombinaciju algoritama za određivanje i povezivanje značajki. Područje ispod krivulje često se koristi za mjerenje učinkovitosti algoritma. Što krivulja leži bliže gornjem lijevom kutu, bolja je učinkovitost.

2.2.1.3.2. Uspješno povezivanje

Jednom kad se odlučimo za strategiju povezivanja, još uvijek trebamo uspješno pronaći potencijalne kandidate za povezivanje. Najjednostavniji način za pronalaženje svih odgovarajućih značajnih točaka je uspoređivanje svih značajki s drugim značajkama u svakom paru potencijalno sparivih slika. Nažalost to je kvadratni broj očekivanoga broja značajki, što ga čini nepraktičnim za većinu aplikacija.

Najbolji pristup je osmisliti strukturu podataka kao što su višedimenzionalna stabla ili tablica raspršivanja za brzo traženje značajki u blizini zadane značajke. Takve se strukture mogu raditi za svaku sliku neovisno ili globalno za sve slike u zadanoj bazi.

Jedna od najjednostavnijih tehnika implementrianja je višedimenzionalno raspršivanje, koje mapira deskriptore u jedinice fiksne veličine bazirane na nekim funkcijama primijenjenim na svaki vektor deskriptora. Prilikom povezivanja, svaka nova značajka smiješta se u jedinicu i pretraga obližnjih jedinica se koristi za odabir potencijalnih kandidata

Druga široko korištena metoda strukturiranja su višedimenzionalna stabla. Najpoznatije od njih je k-d stablo, koje dijeli prostor ravninama. O k-d stablima više će biti govora u poglavlju 6.

Provjera i gustoća povezivanja značajki

Jednom kad imamo hipotetske poveznice, često se koristimo i geometrijskim poravnavanjem kako bismo provjerili koja povezivanja leže a koja ne leže u ravnini. Naprimjer, ako očekujemo da će se cijela slika translatirati i rotirati u pogledu povezivanja, možemo prilagoditi globalnu geometrijsku transformaciju i zadržati samo one povezane značajke koje su dovoljno blizu predviđenoj transformaciji. Proces odabiranja malog seta značajki, a potom provjeravanje većeg seta naziva se nasumično uzorkovanje ili RANSAC. RANSAC će biti detaljnije objašnjen u poglavlju 6.

3. OpenCV

OpenCV je vizijska biblioteka otvorenoga koda. Biblioteka je pisana u programskim jezicima C i C++ i može se izvoditi na svim platformama.

OpenCV je dizajniran za računsku produktivnost s velikim naglaskom na aplikacije u stvarnom vremenu. Jedan od osnovnih ciljeva OpenCV-a je dati jednostavnu uporabu infrastrukture računalnoga vida koja pomaže programerima napraviti sofisticirane vizijske aplikacije u vrlo kratkom roku. Biblioteka OpenCV-a sadržava više od 500 funkcija koje obuhvaćaju mnoga područja računalnoga vida, uključujući inspekciju proizvoda u tvornicama, medicinsku dijagnostiku, osiguranje i nadzor, kalibraciju kamera, stereo-vid i robotiku. Budući da su računalni vid i strojno učenje bliska područja, OpenCV također sadržava i biblioteku strojnoga učenja (*Machine Leraning Library - MLL*). Ta je biblioteka fokusirana na prepoznavanje uzoraka i grupiranje. ML biblioteka je vrlo korisna za vizijske zadatke koji su

jezgra zadaće OpenCV-a, a isto tako dovoljno je dobra kod primjene mnogih zadataka strojnog učenja.

OpenCV je usmjeren na pružanje osnovnih alata koji su potrebni za rješavanje problema računalnoga vida. U nekim slučajevima funkcionalnosti više razine u bibliotekama će biti dovoljne za rješavanje najsloženijih problema računalnoga vida. Pa čak i kad to nije slučaj, osnovne komponente biblioteke dovoljne su za rješavanje gotovo svakog problema računalnoga vida.

3.1. Povijest OpenCV-a

OpenCV je nastao Intelovom inicijativom kako bi se unaprijedili procesori za snažnije programe. Prema tom cilju, Intel je pokrenuo mnoge projekte kao što su praćenja u stvarnom vremenu i 3D zasloni. Osim u Intelu, funkcije koje se danas nalaze u biblioteci OpenCV-a razvijali su i studenti na prestižnim sveučilištima, međusobno ih dijelili i tako ih sve više poboljšavali. Zbog toga je OpenCV postao općenito dostupan svima. Osim Intelovih programera i studenata, na razvoju OpenCV-a su uvelike pomogli i stručnjaci iz Rusije.

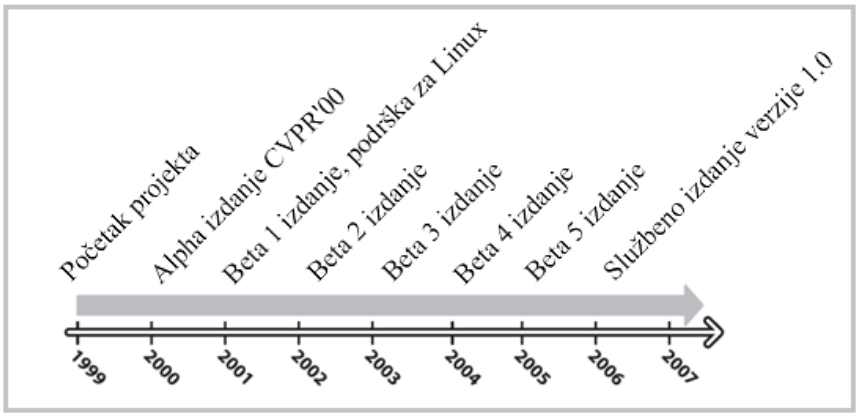
Na početku je bilo postavljeno nekoliko ciljeva OpenCV-a:

- Napredno istraživanje koje daje ne samo dostupne kodove, nego i optimizirane za osnovne vizijske infrastrukture. Jednom riječju, nema više izmišljanja kotača.
- Širenje znanja pružajući jednostavne infrastrukture na kojima razvojni programeri mogu graditi, kako bi se kodovi mogli lakše čitati i prenositi.
- Dostupnost aplikacija računalnoga vida povećava potrebu za bržim procesorima. To je jedan i od razloga zašto je OpenCV besplatan. Prodaja bržih procesora donosi Intelu više zarade nego prodaja softvera.

U razvoju svakog otvorenog koda važno je postići kritičnu masu na kojoj projekt postaje samoodrživ. A OpenCV danas broji milijunske korisnike, koji svakodnevno daju doprinose u razvoju novih i složenijih kodova.

Slika 5. prikazuje vremensku skalu razvoja OpenCV-a. Tijekom razvoja, na OpenCV je utjecao uspon i pad *dot-com-a* kao i mnoge promjene u upravljanju i rukovođenju. Tijekom tih kolebanja bilo je i vremena kad nitko iz Intela nije radio na razvoju OpenCV-a. No,

pojavom višejezgrenih procesora i mnogih novih aplikacija računalnoga vida, vrijednost OpenCV-a je počela rasti. Danas je OpenCV aktivno područje razvoja u nekoliko institucija, stoga se mogu očekivati mnoga poboljšanja u kalibraciji više kamera, dubinskoj percepciji, metodama korištenja lasera zajedno s vizijom, te bolje prepoznavanje uzoraka kao i bolja podrška za robotski vid.



Slika 5. Vremenska skala razvoja OpenCV-a

3.2. Budućnost OpenCV-a

Jedno od ključnih područja razvoja OpenCV-a je robotska percepcija. Tu se fokusira na 3D percepciju, te isto tako i na 2D i 3D prepoznavanje objekata. Robotska percepcija se najviše oslanja na 3D prikazu, stoga se dosta radi na proširivanju kalibracije kamera, usklađivanju kamera i korištenju kombinacija kamere i lasera koji traži udaljenosti.

Uz opažanje 3D objekata, roboti će morati i prepoznati 3D objekte i njihove pozicije. Danas već postoje razvijene funkcije unutar OpenCV-a za jednostavno 2D prepoznavanje objekata. Izrada vještih robota spaja većinu područja računalnoga vida i umjetne inteligencije, od točne 3D rekonstrukcije do praćenja, prepoznavanja osoba, objekata, kao i učenje slika, kontrolu, planiranje i donošenje odluka.

Iako OpenCV nema potpuni fokus na algoritma u stvarnom vremenu, razvijaju se i te tehnike. Nitko ne može reći zasigurno kako će se točno razvijati, ali veliki su prioriteti na sljedeća područja:

3D

Očekuje se bolja podrška za 3D senzore i kombinaciju 2D kamera sa 3D mjernim uređajima. Također se očekuju i bolji algoritmi za stereo-vid kao i podrška za strukturirana osvjetljenja.

Prepoznavanje značajki

Za podršku boljega prepoznavanja značajki mogu se očekivati potpuno opremljeni alati koji će imati okvir za izmjenjivo otkrivanje značajnih točaka. To uključuje poznate algoritme kao što su SUFR, HoG, MSER i ostale.

Infrastruktura

Uključuje bolju podršku za grafičkim sučeljem, poboljšanu dokumentaciju, bolje otkrivanje i otklanjanje pogrešaka kao i podršku za *Linux* sustav.

Sučelje kamere

Bolje upravljanje kamerama s podrškom za kamere s visokim dinamičkim dosegom. Trenutno većina kamera podržava samo 8-bitne kanale, ali novije kamere mogu podržati i 10 ili 12-bitne kanale. Veći dinamički doseg kamera donosi sa sobom i bolje prepoznavanje i stero-vid zato što im je omogućeno otkrivanje suptilnijih tekstura i boja, na što su starije kamere sa slabijim dosegom slijepe.

4. IP KAMERA

Internet protokol kamera ili IP kamera je tip digitalne kamere koja se obično rabi za nadzor i koja, za razliku od analognih CCTV kamera, može primati i slati podatke preko lokalne mreže ili interneta. Iako su većina kamera koje to rade web-kamere, izraz IP kamera obično se rabi za kamere koje se koriste za nadzor. Postoje dvije vrste kamera:

1. Centralizirane IP kamere, koje trebaju centralni mrežni video-rekorder za snimanje.
2. Decentralizirane, koje ne trebaju centralni rekorder jer kamera ima funkcionalnosti ugrađene u sebi.

Na tržištu su dostupne IP kamere s rezolucijama od 0.3 do 20 megapiksela.

Prednosti IP kamera:

- Veća rezolucija, jer IP kamere imaju minimalnu rezoluciju 640x480, a mogu dati i puno veću s HDTV kvalitetom slike i do 30 slika po sekundi.
- Fleksibilnost, može se postavljati bilo gdje na IP mreži, uključujući i bežične mreže.
- Prijenos naredbi za PTZ kamere (kamera koja ima mogućnost rotacije po vertikalnoj i horizontalnoj osi, te mogućnost zumiranja) preko samo jednog kabla
- Šifriranje i provjera, IP kamere nude siguran prijenos podataka šifriranim i sigurnim metodama kao što su WEP.WPA.WPA2.TKIP,AES
- Pristupanje na daljinu, video sa odabrane kamere može se vidjeti na bilo kojem računalu koje se bilogdje nalazi.
- Moderne IP kamere mogu raditi i bez dodatnog napajanja, odnosno rade preko PoE protokola koji daje napajanje kameri preko eternet kabla.

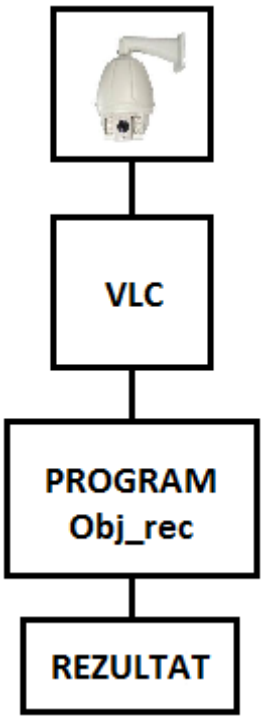
Nedostatci IP kamere

- Previsoka cijena kamere, zbog čega si više koriste obične web kamere
- Potreba za visokom mrežnom propusnošću; tipična CCTV kamera sa rezolucijom od 640x480 piksela u sa 10 slika u sekundi u MJPEG formatu zahtjeva brzinu oko 3Mbit/s.
- Problem oko prenošenja zapisa preko interneta zbog mogućih narušavanja sigurnosti, jer je zapis dostupniji mnogim hakerima.

U Laboratoriju za projektiranje izradbenih i montažnih sustava nalazi se IP IR PTZ kamera, model SN-IPS54/Z26 sa specifikacijama priloženim u prilogu diplomskoga rada. Kameri se pristupa preko softvera u kojemu je omogućeno pomicati kameru u bilo koju poziciju. Pomoću tog softvera dovest ćemo kameru u četiri željene pozicije koje ćemo spremiti pod imenima Preset 10, 11, 12, 13. S tih pozicija će se uzimati scene koje će se spremiti u direktorij na računalu i kasnije obrađivati razvijenim programom za prepoznavanje objekata

5. ZADATAK

Zadatak diplomskoga rada može se podijeliti u dva osnovna koraka. Prvi korak je razvijanje programske podrške za prepoznavanje više različitih objekata u radnom prostoru vizijskim sustavom. Razvijanje programa temelji se na OpenCV biblioteci otvorenoga koda. Drugi dio je povezivanje programskoga rješenja s dostupnom PTZ IP kamerom koja se nalazi u Laboratoriju za projektiranje izradbenih i montažnih sustava na Fakultetu strojarstva i brodogradnje u Zagrebu. Na slici 6. je prikazan pristup problemu i grafički prikaz rješenja zadatka.



Slika 6. Grafički prikaz zadatka

Prvo ću opisati razvijeni program za prepoznavanje objekata, nakon toga povezivanje programa s IP kamerom i na kraju ću prikazati dobivene rezultate pretrage scena, u kojima će se vidjeti nalazi li se objekt ili ne nalazi na sceni.

Objekti koje sam odabrao za svoj zadatak su daljinski upravljač i kutija (slika 7. i slika 8.) Odabrana su ta dva objekta jer se na njima može pronaći mnogo značajki, odnosno ključnih točaka, što dovodi do boljega prepoznavanja objekata. Dakle, što se više značajnih točaka pronađe na slici, veće su šanse za točnijim prepoznavanjem.

Slika 7. Kutija – prvi model koji se traži na sceni

Slika 8. Daljinski upravljač – drugi model koji se traži na sceni

Odabrane scene na kojima će se tražiti objekti prikazane su na slici 9.

(a) – scena na kojoj se nalazi kutija

(b) – scena na kojoj se nalazi daljinski upravljač

(c) – scena na kojoj se nalaze oba objekta

(d) – scena bez objekata

Slika 9. Scene na kojima se traži objekt

6. RAZVOJ PROGRAMSKE PODRŠKE ZA PREPOZNAVANJE VIŠE OBJEKATA

Program za prepoznavanje više objekata napisan je u C++ jeziku koristeći OpenCV biblioteku. Kako je već napomenuto, ta biblioteka sadržava funkcije koje se odnose na područje računalnoga vida. Zadatak prepoznavanja u suštini se može svesti na tri osnovna dijela, odnosno algoritma. Prvi od njih je SURF koji na slici traži značajne točke (blob-strukture), zatim k-NN algoritam za traženje parova točaka (deskriptora) i na kraju homografija kako bi se povezale točke u prostoru. U sljedećim poglavljima detaljnije su opisana tri osnovna algoritma te njihova primjena u programu zadatka diplomskoga rada.

6.1. SURF (*Speed up robust feautre*)

Zadatak traženja sličnosti između dviju slika na istom prizoru ili objektu dio je mnogih vizijskih aplikacija. Kalibracija kamere, 3D rekonstrukcija, registracija slika i prepoznavanje objekata samo su neki od primjera mnoštva vizijskih sustava. Zadatak traženja sličnosti između objekata podijeljen je na tri glavna koraka. Prvo, značajne točke su odabrane na karakterističnim lokacijama slike kao što su kutovi, blobovi i T-spojišta. Najvažnije obilježje detektora značajnih točaka je ponovljivost. Sljedeće, susjedstvo svake značajne točke predstavljeno je sa vektorom. Taj deskriptor mora biti u isto vrijeme karakterističan, robustan na šum, pogreške u pronalaženju i na geometrijske i fotometrijske deformacije. Na kraju, vektori deskriptora se sparuju između različitih slika. Sparivanje je često temeljeno na udaljenosti između dva vektora. Dimenzija deskriptora utječe na vrijeme izvođenja, pa je stoga manji broj dimenzija poželjniji.

6.1.1. Otkrivanje značajnih točaka

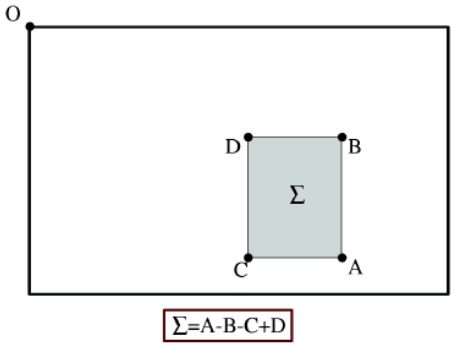
Otkrivanje značajnih točaka koristi osnovnu aproksimaciju Hesseove matrice. To omogućuje korištenje integralnih slika koje drastično smanjuju vrijeme računanja.

6.1.1.1. Integralna slika

Ulaz integralne slike $I_\Sigma(x)$ na lokaciji $x = (x,y)^T$ predstavlja sumu svih piksela ulazne slike *I* u pravokutnom području formiranim s ishodištem i smjerom *x*.

$$I_\Sigma(x) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j)$$

Jednom kad je integralna slika izračunata, uzimaju se tri sume kako bi se izračunala suma intenziteta u pravokutnom području. O veličini pravokutnoga područja ovisi i brzina računanja, pa je poželjno imati što manje područje.



Slika 10. Suma intenziteta u pravokutnom području bilo koje veličine

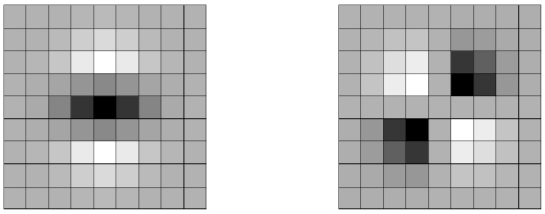
6.1.1.2. Značajne točke bazirane na Hesseovoj matrici

Detektor je baziran na Hesseovoj matrici zbog njezine izrazito dobre karakteristike, a to je točnost. Detektiraju se blob-strukture na lokacijama gdje je determinanta matrice maksimum. Determinanta Hesseove matrice je također bitna i za odabir skaliranja. S obzirom na danu točku *x = (x,y)* na slici *I*, Hesseova matrica *H(x, σ)* u točki *x* na skali *σ* je definirana na sljedeći način

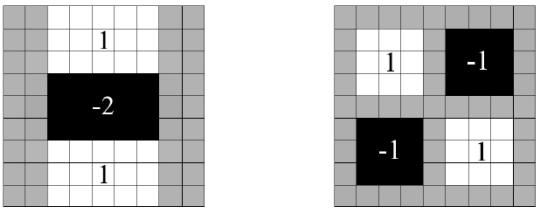
$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix},$$

gdje je $L_{xx}(x,\sigma)$ konvolucija derivacije drugog reda Gaussove funkcije na slici *I* u točki *x*, slično vrijedi i za $L_{xy}(x,\sigma)$ i $L_{yy}(x,\sigma)$.

Derivacije Gaussove funkcije su optimalne za analizu skaliranja prostora (slika 11.), ali u praksi moraju biti diskretizirane i pojednostavljene (slika 12.). To dovodi do gubitka ponovljivosti pri rotaciji slike oko neparnih faktora $\frac{\pi}{4}$. Ovaj nedostatak općenito vrijedi za detektore bazirane na Hesseovoj matrici.



Slika 11. Derivacije Gaussove funkcije u x i xy smjeru



Slika 12. Aproksimirane derivacije Gaussove funkcije u x i xy smjeru

Filtri koji su nazvani box-filtri veličine su 9x9 i aproksimacija su derivacija Gaussove funkcije sa $\sigma = 1.2$, te predstavljaju najmanju skalu za računanje odziva blob-mapa. Označeni su sa D_{xx}, D_{yy}, D_{xy}. Težine koje se odnose na pravokutna područja zadržavaju jednostavnost pri računanju. Time se dobiva:

$$\det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2$$

Relativna težina w odziva filtra koristi se za uravnoteženje izraza determinante Hesseove matrice. To je potrebno zbog očuvanja energije između jezgre Gaussove funkcije i jezgre aproksimirane Gaussove funkcije,

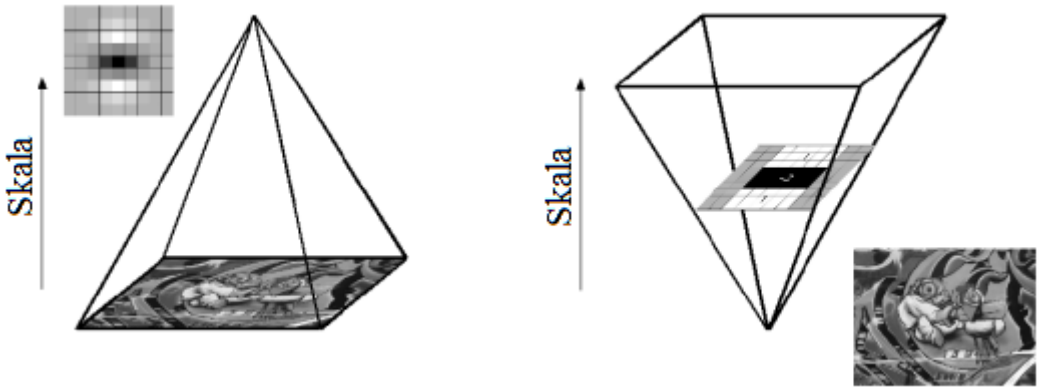
$$w = \frac{|L_{xy}(1.2)|_F |D_{yy}(9)|_F}{|L_{yy}(1.2)|_F |D_{xy}(9)|_F} = 0.912 \ldots \cong 0.9$$

gdje je $|x|_F$ Forbeniusov standard. Za teoretsku točnost težina se mijenja u ovisnosti o skali. U praksi faktor je konstantan budući da nema neke značajne utjecaje u dobivenim rezultatima. Nadalje, odzivi filtara su normalizirani u odnosu na njihovu veličinu. To garantira konstantan Forbeniusov standard za bilo koju veličinu filtra, što je bitan aspekt za analizu prostora skala.

6.1.1.3. Prikaz prostora skala

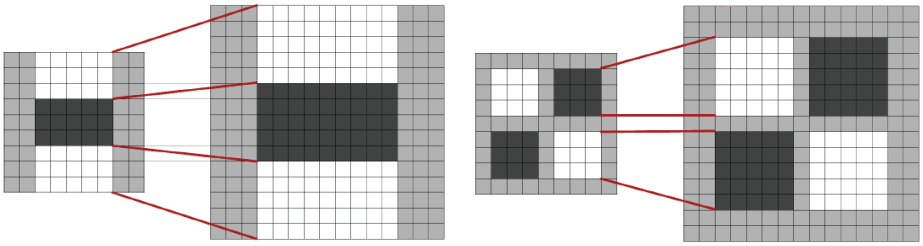
Značajne točke treba naći u različitim mjerilima zato što pretraga sličnosti često treba njihovu usporedbu na slikama gdje se vide u različitim mjerilima. Prostor skala općenito je implementiran kao piramida slike. Slike se u više navrata izglađuju Gaussovim funkcijama, a potom se smanjuju kako bi postigle višu razinu piramide.

Zbog uporabe box-filtra i integralnih slika nije potrebno iterativno primjenjivati isti filtar na izlaz prethodno filtriranoga sloja, umjesto toga može se primijeniti box-filtar bilo koje veličine s istom brzinom izravno na originalnu sliku. Dakle, prostor skala se analizira povećanjem veličine filtra umjesto iterativnim smanjenjem veličine slike (slika 13.). Izlaz filtra veličine 9x9 definiran je kao početna razina skale piramide koji se odnosi na skalu $s=1.2$ (aproksimacija derivacije Gaussove funkcije sa $\sigma=1.2$). Sljedeće se razine piramide dobivaju filtriranjem slike s postupno većim maskama, uzimajući u obzir diskretnu narav integralne slike i specifičnu strukturu filtra.



Slika 13. Piramidalni prikaz povećanja veličine filtra

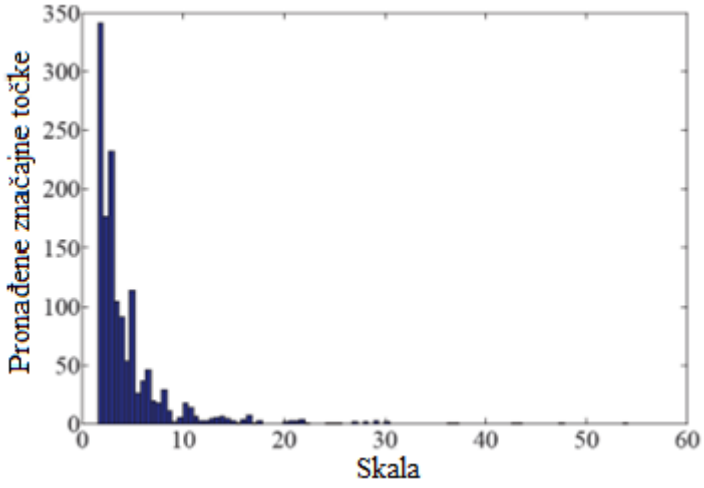
Prostor skala je podijeljen u dvije oktave. Oktava predstavlja niz odziva filtra koji su dobiveni konvolucijom iste ulazne slike povećanjem veličine filtra. Svaka je oktava podijeljena na jednak broj skala. Minimalna razlika između dvije iduće skale ovisi o duljini l_0. Ta filtar 9x9 duljina l_0 je 3. Za dvije sljedeće razine veličina se mora povećati za minimalno 2 piksela (po jedan piksel sa svake strane) kako bi se zadržala neparnost i time osigurala prisutnost centralnoga piksela. To rezultira ukupnim povećanjem maske za 6 piksela. Slika 14. prikazuje povećanje filtra D_{xx} i D_{yy} sa 9x9 na 15x15 veličinu u y i xy smjeru. Povećati se može za samo paran broj piksela.



Slika 14. Povećanje filtra u y i xy smjeru

Konstrukcija prostora skala počinje sa filtrom veličine 9x9, koji računa odaziv bloba na slici za najmanju skalu. Nakon toga se primjenjuju redom filtri veličina 15x15, 21x21, 27x27. Nakon interpolacije najmanja moguća skala je $\sigma = 1.6 = 1.2\frac{12}{9}$ s odgovarajućim filtrom veličine 12x12, a najveća skala $\sigma = 3.2 = 1.2\frac{24}{9}$.

Isto se razmatranje uzima i za ostale oktave. Za svaku novu oktavu veličina filtra se podvostručuje. U isto vrijeme i intervali uzorkovanja za određivanje značajnih točaka se mogu poduplati za svaku novu oktavu. To smanjuje vrijeme računanja i gubitak na točnosti. Veličine filtara druge oktave su 15, 27, 39, 51. Treća oktava se računa sa filtrima veličine 27, 51, 75, 99. Ako je originalan slika još uvijek veća od odgovarajućeg filtra, analizira se i četvrta oktava primjenom filtra veličina 51, 99, 147 i 195. Bitno je napomenuti da što se veći broj skala analizira, brže opada broj pronađenih značajnih točaka (slika 15.).



Slika 15. Histogram detektiranih skala

6.1.1.4. Lokalizacija značajnih točaka

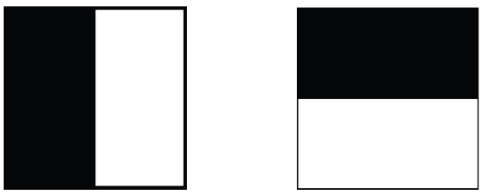
Kako bi se lokalizirale značajne točke na slici i kroz skale, primjenjuje se odbacivanje svih točaka koje nisu maksimum determinante Hesseove matrice a koje se u području 3x3x3. Nakon toga se maksimum determinante Hesseove matrice interpolira u prostor skale i slike.

6.1.2. Opisivanje značajnih točaka

Deskriptor opisuje raspodjelu intenziteta unutar susjedstva značajnih točaka. Temelji se na raspodjeli prvoga reda odziva Haarovih oscilacija u *x* i *y* smjeru, koristeći se pri tome integralnom slikom za brzinu i samo 64 dimenzije. To smanjuje vrijeme računanja i uspoređivanja značajki i dokazano je da u isto vrijeme povećava i robusnost. Za opisivanje se koriste i indeksiranja na temelju Laplaceovog znaka, što ne samo da povećava robusnost deskriptora nego povećava i brzinu uspoređivanja (*matching*). Prvi se korak sastoji od određivanja orijentacije bazirane na informacijama kružnih polja oko značajne točke. Nakon toga radi se pravokutno područje usklađeno s odabranom orijentacijom iz kojega se određuje SURF deskriptor.

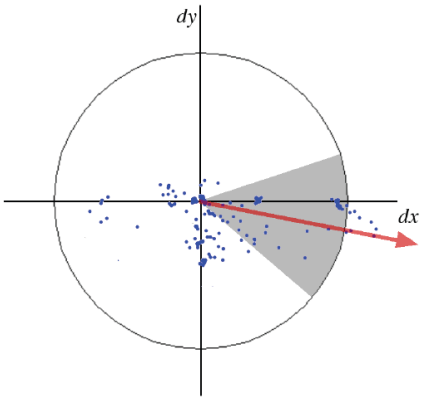
6.1.2.1. Određivanje orijentacije

Kako bi bile neovisne o rotaciji slike, određuju se orijentacije za značajne točke. Kako bi se to izvelo, potrebno je izračunati odziv Haarovih oscilacija u *x* i *y* smjeru unutar kružnoga područja radijusa *6s* oko značajne točke sa skalom u kojoj je značajna točka pronađena. Korak uzorkovanja *s* ovisi o skali. Kako bi se zadržala svojstvenost, oscilacije su također neovisne o skali i postavljene na veličinu *4s*. Stoga se ponovno mogu koristiti integralne slike za brzo filtriranje. Na slici 11. prikazani su filtri koji računaju odzive u *x* i *y* smjeru. Tamni dio ima težinu -1, a bijeli +1.



Slika 16. Filtri Haarovih oscilacija

Jednom kad je odziv oscilacije izračunat i značajnoj točki pridodana težina s Gaussovom funkcijom *(σ=2s)*, odzivi su predstavljeni točkama u prostru, odnosno koordinatnom sustavu gdje je horizontala snaga odziva na apscisi, a vertikala snaga odziva na ordinati. Dominantna orijentacija se procjenjuje računanjem suma svih odziva unutar kliznoga orijentacijskoga prozora veličine $\frac{\pi}{3}$ (slika 17.).



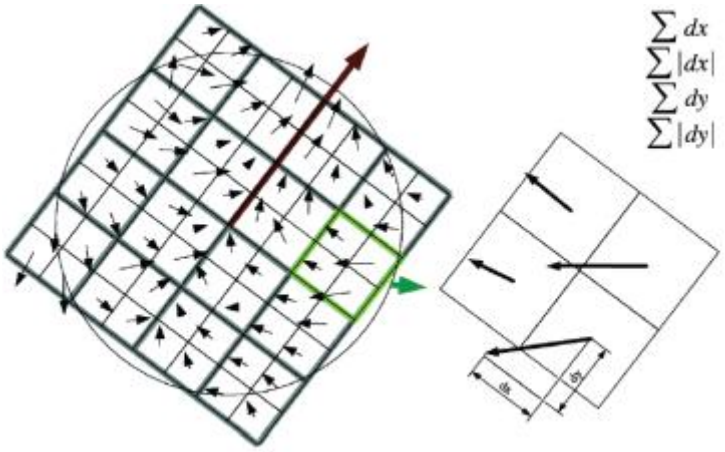
Slika 17. Suma odziva unutar kliznoga orijentacijskoga prozora

Horizontalni i vertikalni odziv unutar prozora se zbrajaju. Suma ta dva odziva predstavlja lokalni orijentacijski vektor. Najduži takav vektor unutar svih prozora definira orijentaciju značajne točke. Veličina kliznoga prozora je parametar koji se pažljivo mora odabrati. Manje veličine prozora određuju jedan dominanti gradijent, a veći prozori imaju tendenciju da daju maksimum u veličini vektora koji nije ni iskazan. U oba slučaja dolazi do pogrešnih rezultata u određivanju orijentacije značajnih točaka.

6.1.2.2. Deskriptor temeljen na sumi odziva Haarovih oscilacija

Prvi korak ekstrakcije deskriptora sastoji se od određivanja kvadratnoga područja smještenoga oko značajne točke i orijentiranoga uzduž orijentacije koja je određena u prijašnjem poglavlju. Veličina tih prozora (područja) je *20s*. Područje je podijeljeno u manja 4x4 područja. To čuva važne prostorne informacije. Za svako to manje područje računa se odziv Haarovih oscilacija na 5x5 regularno udaljenih uzoraka točaka. Zbog jednostavnosti, odziv u horizontalnom smjeru nazvan je d_x a odziv u vertikalnom smjeru *dy*. Kako bi se povećala robusnost prema geometrijskim deformacijama i lokalnim pogreškama, prvo se odzivima d_x i d_y pridodaje težina s Gaussovom funkcijom *(σ=3.3s)* centriranim na značajnoj točki. Nakon toga odzivi d_x

i d_y se zbrajaju za svako područje i formiraju prvi set ulaza za vektor značajke. Slika 18. prikazuje računanje odziva za svaki kvadrat. 2x2 podregije svakoga kvadrata odgovaraju stvarnom polju deskriptora. Za računanje se uzima relativna vrijednost odziva $|d_x|$ i $|d_y|$.



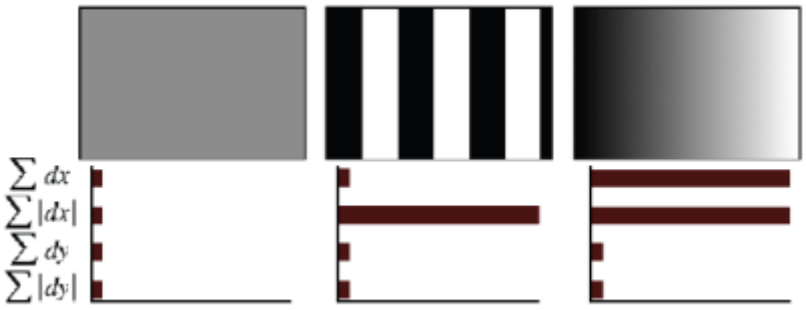
Slika 18. Računanje odziva kvadratne podregije veličine 2x2

Dakle, svaka podregija ima 4 dimenzijska vektora deskriptora *v* za svoju temeljnu strukturu.

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$$

Povezivanjem toga sa svim 4x4 podregijama rezultira veličinom vektora deskriptora 64. Odzivi oscilacija ne ovise o odstupanjima u osvjetljenju. Neovisnost prema kontrastu postignuta je jediničnim vektorom deskriptora.

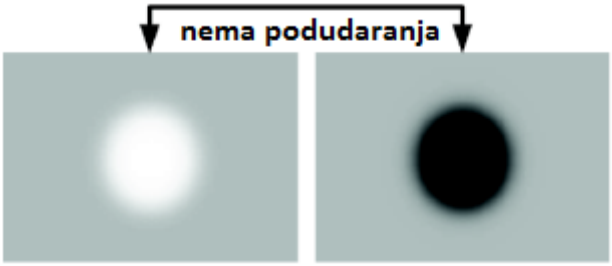
Slika 19. prikazuje svojstva deskriptora za tri različita intenziteta uzoraka u smjeru x na slici unutar podregija. U slučaju homogene regije sve vrijednosti su relativno male, za frekvencijsku regiju u smjeru x vrijednost sume $|d_x|$ je najveća, a ostale su male, a ako se intenzitet postupno povećava u smjeru x, vrijednosti sume *dx* i |*dx*| su velike.



Slika 19. Svojstvo deskriptora kod različitih intenziteta

6.1.2.3. Brzo indeksiranje

Za brzo indeksiranje za vrijeme povezivanja točaka uključuje se predznak Laplacea. Tipično, značajne točke su pronađene kod blob-struktura. Laplaceov znak raspoznaje blobove na svijetloj i tamnoj podlozi. Za to nije potrebno neko dodatno računanje jer je već izračunato tijekom faze detekcije. Kod povezivanja točaka mogu se dakle uspoređivati značajke koje imaju isti predznak (slika 20.). To poboljšava povezivanje točaka bez smanjenja svojstava deskriptora.



Slika 20. Uspoređivanje točaka različitoga Laplaceovoga predznaka

6.2. Primjena SURF- algoritma na zadatak

Unutar OpenCV biblioteke nalazi se funkcija *cvExtractSURF*. Funkcija traži robusne značajke na slici. Za svaku značajku vraća njezinu lokaciju, veličinu, orijentaciju i deksriptor. Koristi se pri pronalaženju objekata. Struktura funkcije glasi:

```
void cvExtractSURF(
        const CvArr* image,
        const CvArr* mask,
        CvSeq** keypoints,
        vSeq** descriptors,
        CvMemStorage* storage,
        CvSURFParams params)
```

Gdje su ulazni parametri:

image – ulazna 8-bitna crno-bijela slika

mask – ulazna 8-bitna maska značajke koje su samo pronađene u području koje sadržava 50% „*non-zero*“ *mask* piksela.

Izlazni parametri:

descriptors – dvostruki pokazivač na niz deskriptora. Ovisno o vrijednosti parametra svaki deksriptor u nizu može biti vektor sa 64 ili 128 elemenata

keypoints – dvostruki pokazivač na niz u kojemu se nalaze ključne točke. Struktura točaka *CvSURFPoint* glasi:

```
typedef struct CvSURFPoint
{
CvPoint2D32f pt; // pozicija značajke unutar slike
int laplacian;   // -1, 0 or +1. Znak Laplacea u točki. Može se iskoristiti za bolje povezivanje točaka (značajke sa različitima Laplaceovim znakom se obično ne povezuju)
int size;        // veličina značajke
float dir;       // orjenzacija značajke: od 0 do 360 stupnjeva
float hessian;   // vrijednost hessiana (može es koristiti za procjenu jačina značajki)

}
CvSURFPoint;
```

Dodatni parametri su:

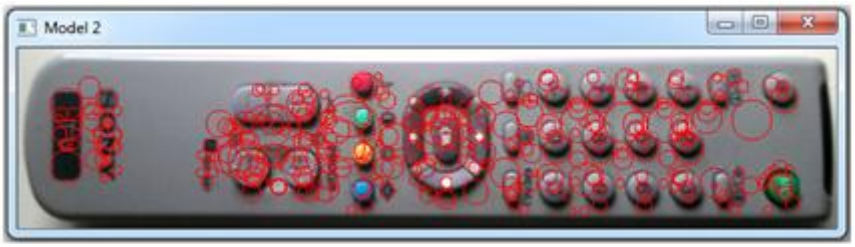
storage – memorijsko spremište gdje se spremaju parametri točaka

params – parametri algoritma koji se stavljaju u strukturu *CvSURFParams*. Ta struktura glasi:

```
typedef struct CvSURFParams
{
int extended; // 0 znači osnovni deskriptor (64 elementa),
              // 1 znači prošireni deskriptor (128 elementa)
double hessianThreshold; // izdvajaju se samo značajke koje imaju vrijednost veću od granice
                         // najbolja vrijednost je između 300 i 500 (može ovisiti o lokalnom kontrastu i oštirni slike)
int nOctaves; // broj oktava koje se koriste pri izdvajanju, sa svakom sljedećom oktavom veličina značajke se podvostručuje
int nOctaveLayers; // broj slojeva unutar svake skale
}
CvSURFParams;
```

Budući da zadatak pronalazi i prepoznaje više objekata, funkcija *cvExtractSurf* primjenjuje se više puta. U program se prvo učitavaju slike dvaju modela objekata koji će se kasnije uspoređivati sa scenama na kojima je potrebno pronaći i prepoznati objekt. Učitavanje slika odrađuje *cvLoadeImage* funkcija kojoj je potrebno dati putanju do tražene slike. Pošto je navedeno da je ulazni parametar funkcije *cvExtractSurf* crno-bijela 8-bitna slika, učitanu sliku potrebno je obraditi. Sliku prvo pretvaramo u 8-bitnu sliku pomoću funkcije *cvCreateImage,* nakon toga funkcijom *cvCvtColor* pretvaramo sliku u crno-bijelu. Kada imamo željenu ulaznu sliku, već rečenom funkcijom *cvExtractSurf* izdvajamo ključne točke i deskriptore. Slike 21. i 22. prikazuju objekte daljinski upravljač i kutiju s pronađenim značajkama odnosno blob-strukturama.

Slika 21. Pronađene blob-strukture na kutiji



Slika 22. Pronađene blolb-strukture na daljinskom upravljaču

Istim postupkom pronalaženja značajki na objektima koji je opisan pristupa se i pronalaženju značajki na scenama koje su dobivene s IP kamere. Učitavanje i spremanje slika s kamere bit će opisano u poglavlju 7. Nakon učitavanja i pronalaženja značajki na scenama slijedi algoritam koji povezuje značajne točke, odnosno njihove deskriptore.

6.3. NN search

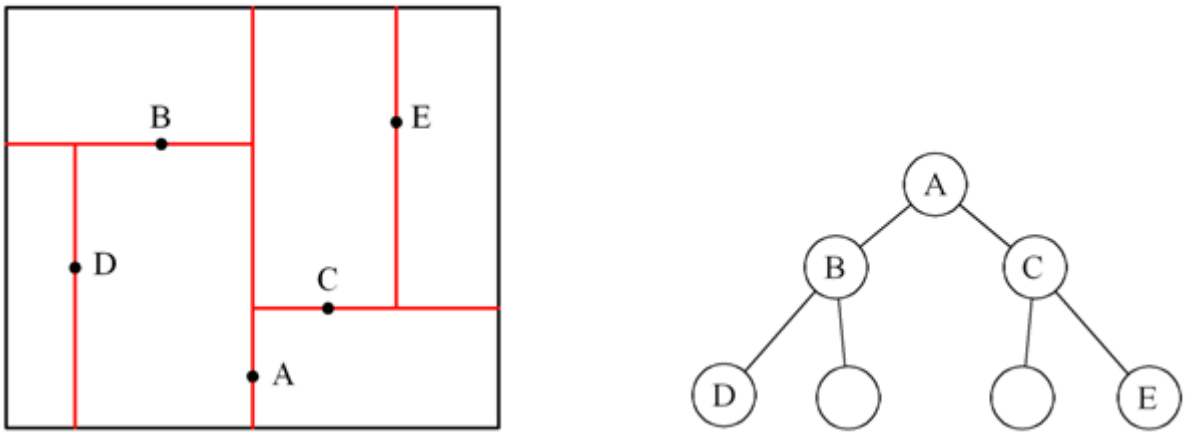
NN(*nearest neighbor*) *search* ili pretraga najbližih susjeda je optimizacijski problem za traženje najbližih točaka u metričkom prostoru. Problem glasi: ako je zadan set točaka *S* u metričkom prostoru *M* i tražena točka *q*, treba naći najbližu točku iz seta točaka *S* koja je najbliža točki *q*. Postoji više algoritama za NN problem. U programu zadatak diplomskoga rada kotisti se k-nn algoritam i struktura podataka *k-d tree* (k-d stablo).

6.3.1. k-d stablo

k-d (*k-dimensional) tree* ili k-dimenzionalno stablo je prostorno raspodijeljena struktura podataka za organiziranje točaka u k-dimenzionalnom prostoru. k-d stabla su korisne strukture podataka za više aplikacija kao što su traženja točaka koja uključuju višedimenzionalna traženja npr. *range serach* i *NN search*.
k-d stablo je binarno stablo u kojemu je svaki čvor k-dimenzionalna točka. Svaki čvor generira ravninu koja dijeli prostor na pola. Točke lijevo od ravnine predstavljene su lijevim podstablom, a točke desno od ravnine desnim podstablom. Pravac ravnine koja dijeli prostor bira se tako da je svaki čvor na stablu povezn s jednom od k-dimenzija s ravninom okomitom na os dimenzije.

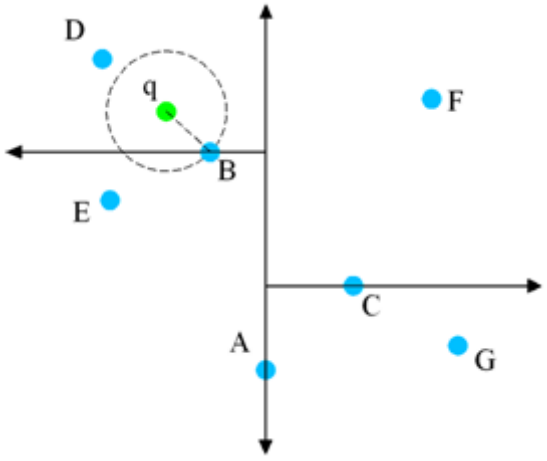
6.3.1.1. Konstrukcija k-d stabla

Za zadan set od n točaka u d-dimenzionalnom prostoru k-d stablo se konstruira na sljedeći način. Prvo se nađe sredina vrijednosti *i*-tih koordinata točaka (u početku *i*=1). Računa se vrijednost *M*, tako da najmanje 50% točaka imaju svoju *i*-tu koordinatu veću ili jednaku *M* a 50% točaka ima manju ili jednaku vrijednosti *M*. Vrijednost *x* je spremljena i set točaka *P* je podijeljen na P_L i P_R, gdje P_L sadržava točke sa *i*-tom koordinatom manjom ili jednakom od *M*. Proces se nakon toga ponavlja rekurzivno na P_L i P_R prostoru. Rezultirajuća struktura je binarno k-d stablo sa *n* listova i dubinom log_n.
Slika 23. prikazuje primjer rezultata podjele prostora i rezultirajuće k-d stablo.



Slika 23. Podjela prostora i rezultirajuće k-d stablo

Jednom kad se strukturiraju podatci, može se krenuti s traženjem. Cilj NN algoritma je naći točku na stablu koja je najbliža ulaznoj točki. k-d stablo ubrzava proces pretrage, jer brzo eliminira velike dijelove prostora. Primjer NN pretrage uz prethodno strukturirane podatke na k-d stablu prikazan je na slici 24.



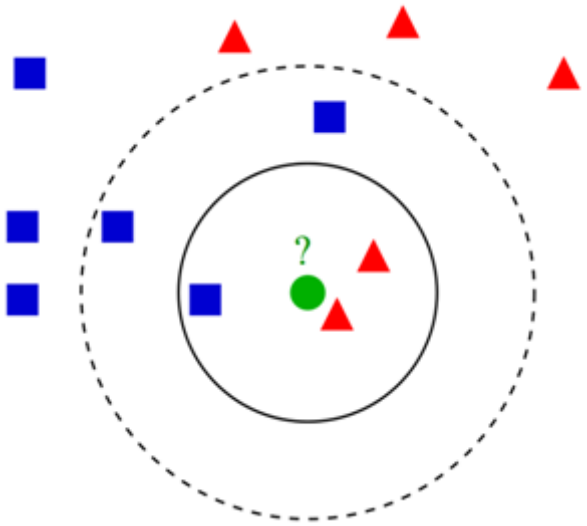
Slika 24. NN pretraga na strukturiranom k-d stablu

Točka *q* se nalazi u setu točaka. Strukturiranjem k-d stabla odmah se mogu isključiti sve točke koje se nalaze u desnom prostoru od točke *A*, čime se smanjuje računanje za svaku točku. Na slici se vidi da je najbliži susjed točka *B* te tražena točka *q* poprima njezine vrijednosti.

6.3.2. k-NN algoritam

k-NN (*k – nearest neighobr*) je metoda klasificiranja objekata na najbližem uzorku učenja u prostoru značajke. k-NN je tip lijenog algoritma za učenje gdje je funkcija aproksimirana samo lokalno i sve računanje je odgođeno do klasifikacije. k-NN je jedan od najjednostavnijih algoritama za učenje. Objekt je određen većinom glasova njegovih susjeda i pridružuje se najvećem broju *k* najbližih susjeda. *k* je pozitivan cijeli broj, tipično je mali broj. Ako je *k=1,* objekt se pridružuje prvom najbližem susjedu po Euklidovoj udaljenosti. Jednom riječju, ako je *k=1,* imamo samo NN algoritam.

Prednost k-NN algoritma je što se njemu pridružuje vrijednost prosjeka većeg broja najbližih susjeda. Susjedi se uzimaju iz seta objekata za koji je poznata točna klasifikacija, npr. točke strukturirane u k-d stablo. Na slici 25. prikazana je shema rada k-NN algoritma.



Slika 25. Princip rada k-NN algoritma

Ako pogledamo zelenu točku u području za *k=3* (područje u kojem se nalaze 3 objekta), ona će poprimiti vrijednosti crvenog trokuta jer broj crvenih trokuta dominira u zadanom području. Ako povećamo vrijednost *k* na *k=5*, oko zelene točke bit će 5 susjeda od kojih će ovaj puta većinu predstavljati plavi kvadrati, pa će zelena točka poprimiti vrijednosti plavoga kvadrata.

6.4. FLANN

U sklopu zadatka koristit će se *FLANN* biblioteka koja sadržava zbir algoritama optimiziranih za traženje najbližega susjeda. Za pretragu ću koristiti *flannFindPairs* funkciju koja ima strukturu:

```
void flannFindPairs(
        const CvSeq*,
        const CvSeq* objectDescriptors,
        const CvSeq*,
        const CvSeq* imageDescriptors,
        vector<int>& ptpairs )
```

Ulazni parametri su deskriptori ulazne slike objekta i objekta na sceni i izlazni parametar parovi točaka. Unutar *flannFindPairs* funkcije nalazi se k-d *tree* algoritam koji će prvo strukturirati podatke. Za to ću koristiti funkciju *cv::cv::flann::Index_<T>::Index* koja u sebi sadržava algoritam za konstruiranje k-d stabla. Struktura je:

Index_<T>::Index_(const Mat& *features*, const IndexParams& *params*)

Gdje je parametar funkcije *params*:

KDTreeIndexParams: konstruira se set slučajno odabranih k-d stabala koji će se paralelno pretraživati. Struktura *KDTreeIndexParams* glasi:

```
struct KDTreeIndexParams : public IndexParams
{
   KDTreeIndexParams( int trees = 4 );
};
```

trees je broj paralelnih k-d stabala. Najbolje vrijednosti su u rasponu od 1 do 16.

Strukturiranjem podataka prelazi se na pretragu najbližega susjeda, za to se koristi *knnSearch* funkcija koja ima svoju strukturu:

```
void Index_<T>::knnSearch(
const Mat& queries,
Mat& indices,
Mat& dists,
int knn,
```

const SearchParams& *params*)

gdje su parametri funkcije:

- *query* – tražena točka
- *indices* – vektor koji sadržava indekse k-b najbližih pronađenih susjeda. Mora imati najmanje knn veličinu
- *dists* – vektor koji sadrži udaljenost k-najbližih pronađenih susjeda
- *knn* – broj najbližih susjeda za tražene
- *params* – paametri pretrage

Struktura SearchParams glasi:

```
struct SearchParams {
    SearchParams ( int checks = 32);
};
```

gdje je *cheks* broj rekurzivnih prolaza kroz koje stablo prolazi. Veća vrijednost za ovaj parametar će dati bolju preciznost pretraživanja, ali za to će biti potrebno i više vremena.

Nakon određivanja parova objekta i objekta na sceni potrebno je još izračunati homografiju kao bi se povezale kutne točke koje će povezene linijom zaokruživati pronađeni objekt na slici ako bude dovoljno povezanih parova deskriptora.

6.5. Homografija

2D točka *(x,y)* na slici može se prikazati kao 3D vektor $x = x_1, x_2, x_3$ gdje je $x = \frac{x_1}{x_3}$, a $y = \frac{x_2}{x_3}$.
To se zove homogeno prikazivanje točke i leži na projiciranoj ravnini P_2. Homografija je inverzno mapiranje točaka i linija na projiciranoj ravnini P_2. Još jedna definicija homografije kaže da je homografija takvo inverzno mapiranje točaka s ravnine P_2 u samu sebe kod kojega tri točke leže na istoj liniji ako i samo ako su njihove mapirane točke kolinearne.
Teorem na kojem je zasnovana algebarska definicija glasi: *mapiranje iz ravnine P_2 u P_2 je projiciranje samo ako postoji ne-singularna matrica H 3x3 takva da za svaku točku u P_2 predstavljenu vektorom x vrijedi da je mapirana točka jednaka umnošku Hx*. To znači, kako bi izračunali homografiju koja mapira svaki x_i u odgovarajući x'_i dovoljno je izračunati 3x3

matricu homografije *H*. Homografije između slika procjenjuju se traženjem odgovarajućih značajki na slikama. Najčešće korišteni algoritam koristi odgovarajuće točke značajki iako mogu biti korištene i ostale značajke, primjerice linije.

6.5.1. Veza s drugim geometrijskim transformacijama

Dobar način da se razumije homografija jest povezivanje homografije s kontekstom drugih geometrijskih transformacija. Homografske transformacije imaju 8 stupnjeva slobode ali postoje i druge transformacije koje i dalje koriste 3x3 matricu, ali sadržavaju specifična ograničenja kako bi smanjile broj stupnjeva slobode. Homografija se može raščlaniti na jednostavnije transformacije.

6.5.1.1. Izometrija

Izometrija je transformacija koja zadržava Euklidovu udaljenost. To znači da će udaljenost između dvije točke na jednoj slici imati istu udaljenost između njihovih odgovarajućih točaka na mapiranoj slici. Isto se odnosi i na kutove između linija. Izometrije sačinjavaju samo 2D rotacije i 2D translacije i zbog toga imaju 3 stupnja slobode. Izometrija se može zapisati kao:

$$x' = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} x,$$

gdje *R* označava 2x2 rotacijsku matricu, *t* je translacijski vektor a *0* je red od dvije nule.

6.5.1.2. Transformacija sličnosti

Transformacija sličnosti je slična izometriji osim što sadržava i izotropsko skaliranje. Izotropsko znači da je skaliranje invarijantno s očuvanjem smjera. Skale dodaju dodatni stupanj slobode pa transformacija sličnosti sadržava ukupno 4 stupnja slobode. Isto kao i kod izometrije, kutovi nisu pod utjecajem transformacije. Udaljenost između točaka više nije invarijantna, ali se zadržava omjer udaljenosti. Transformacija sličnosti se može zapisati kao:

$$x' = \begin{pmatrix} sR & t \\ 0^T & 1 \end{pmatrix} x,$$

gdje je *s* skalar i predstavlja izotropsko skaliranje.

6.5.1.3. Affine transformacije

Affine transformacije su kao transformacije sličnosti, ali umjesto jedne rotacije i izotropskoga skaliranja sastoje se od dvije rotacije i dva ne-izotropska skaliranja. Sadržavaju također i dodatna dva stupnja slobode, jedan za kut koji određuje smjer skaliranja i jedan za omjer parametara skaliranja. Za razliku od transformacije sličnosti, Affine transformacije ne zadržavaju omjer udaljenosti kutova između linija. Affine transformacije se mogu zapisati kao:

$$x' = \begin{pmatrix} A & t \\ 0^T & 1 \end{pmatrix} x,$$

gdje je *A* 2x2 ne-singularna matrica.
Matrica *A* se može raščlaniti i zapisati kao:

$$A = R(\theta)R(-\emptyset)dr(\emptyset),$$

Gdje su *R(θ)* i *R(∅)* rotacijske matrice za *θ* i ∅, a D je dijagonalna matrica:

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

λ_1 i λ_2 su dvije vrijednosti skaliranja.
Stoga je matrica *A* povezuje rotaciju za ∅, skaliranje za λ_1 u smjeru *x*, skaliranje za λ_2 u smjeru *y* i rotaciju unatrag za -∅, te još jednu rotaciju za *θ*.

6.5.1.4. Projicirana transformacija

Projicirana transformacija ili homografije je ne-singularna linearna transformacija homogenih koordinata. Projicirana transformacija sadržava dodatna dva stupnja slobode u odnosu na Affine transformacije. Projicirana transformacija se može zapisati kao:

$$x' = \begin{pmatrix} A & t \\ v^T & v \end{pmatrix} x$$

gdje je $v = (v_1, v_2)^T$.

Ključna razlika između Affinih i projiciranih transformacija je vektor *v*, koji je 0 kod Afinih transformacija. Taj vektor je odgovoran za nelinearne učinke projiciranja. Kod Affinih transformacija skaliranja iz *A* su jednaka svugdje u ravnini, a kod projiciranih skaliranja

variraju s pozicijom u slici. Slično tomu, kod Affinihi transformacija, orijentacija transformirane linije ovisi samo o orijentaciji originalne linije dok kod projicirane transformacije i pozicija originalne slike utječe na orijentaciju transformirane linije. Projicirane transformacije se mogu raščlaniti u lanac prethodno spomenutih transformacija:

$$H = H_S H_A H_P = \begin{pmatrix} sR & t \\ 0^T & 1 \end{pmatrix} \begin{pmatrix} U & 0 \\ 0^T & 1 \end{pmatrix} \begin{pmatrix} \mathrm{I} & 0 \\ v^T & v \end{pmatrix} = \begin{pmatrix} A & t \\ v^T & v \end{pmatrix}$$

6.5.2. Primjena homografije

Postoje mnoge situacije u primjeni računalnoga vida gdje je potrebno procjenjivati homografiju. Neka važnija područja primjene homografije su:

- Kalibracija kamere
- 3D rekonstrukcija
- Vizualno mjerenje
- Stereo vid
- Razumijevanje scene

6.5.3. RNASAC (Random Sample Consensus)

U programu za prepoznavanje objekata koristi se homografija sa *RANSAC* algoritmom.
Princip rada algoritma je sljedeći:

1. Prvo moraju biti pronađeni krajnji kutovi na obje slike.
2. Primjenjuju se normalizirane korelacije između kutova i parovi s dovoljno velikim rezultatom podudaranja tvore kandidate povezivanja.
3. 4 točke se odabiru iz seta kandidata i računa se homografija
4. Odabiru se parovi koji se slažu sa homografijom
5. Ponavljaju se koraci 3. i 4. dok je dovoljan broj parova u skladu s izračunatom homografijom.

6.5.3.1. *Odabiranje 4 točke*

Način na koje su odabrane četiri točke u trećem koraku algoritma utjecat će na točno određivanje homografije. Zasad se točke uzimaju slučajnim odabirom iz seta kandidata. Područje četiri trokuta određenih točkama se računa i točke se uzimaju za sljedeći korak ako je područje veće od zadanoga *tresholda*. Drugi način odabiranja točaka je na temelju istih rubova na slici. Točke koje se nalaze na istom rubu imaju veću vjerojatnost da su kolinearne ili da se nalaze blizu jedna drugoj. Rješenje se sastoji od odabira 4 točke na dva različita ruba. To povećava vjerojatnost da se točke nalaze na istoj ravnini. Slično tomu mogu se odabrati i dvije točke koje se nalaze na istom pravcu. To je pogotovu važno kada razlika između dva pogleda postaje značajnija.

6.6. Primjena homogafaije u zadatku

Unutar OpenCV biblioteke nalazi se funkcija *cvFindHomography* koja traži perspektivne transformacije između dviju ravnina, odnosno između izvorne i ciljne ravnine.

$$S_i \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Struktura funkcije *cvFindHomography* glasi:

```
void cvFindHomography(
        const CvMat* srcPoints,
        const CvMat* dstPoints,
        CvMat* H
        int method=0,
        double ransacReprojThreshold=0,
        CvMat* status=NULL)
```

Parametri funkcije su:

srcPoints: koordinate točke na izvornoj ravnini, 2xN, Nx2, 3xN ili Nx3 1-kanalni niz. N je broj točaka

dstPoints: koordinate točaka ciljne ravnine 2xN, Nx2, 3xN ili Nx3 1-kanalni niz

H: izlazna 3x3 matrica homografije
method: metoda za računanje matrice homografije. Metode su:

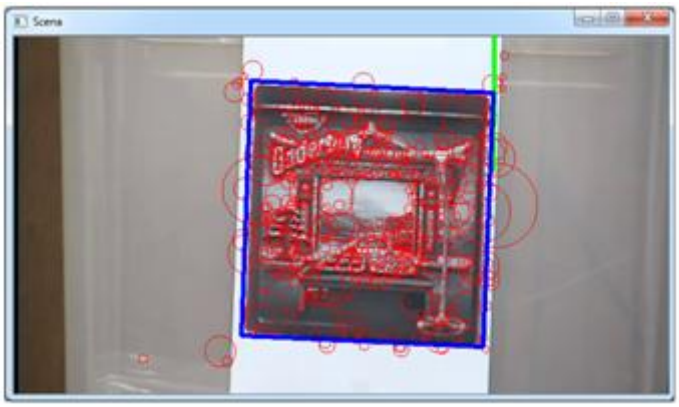
- **0** – obična metoda koja koristi sve točke
- **CV_RANSAC** - RANSAC-metoda
- **CV_LMEDS** - Least-Median robusna metoda

ransacReprojThreshold: maksimalni dopušteni broj grešaka ponovnoga projiciranja (koristi se samo kod EANSAC metode)
status: pozicionalna izlazna maska namještena metodama CV_RANSAC ili CV_LMEDS

Opisom tri osnovna algoritma opisan je ujedno i program prepoznavanja objekata. Dakle još jednom da sumiram, prvo se pomoću SURF algoritma na objektu i sceni traže ključne točke, odnosno blob-strukture, zatim se pomoću k-NN algoritma traže parovi ključnih točaka i na kraju homogrfijom se povezuju kutne točke izvorne slike objekta s objektom na sceni. Ostaje još problem prepoznavanja objekta na sceni, odnosno određivanje kad se objekt nalazi na sceni a kad ga nema.

6.7. Prepoznavanje objekta

Budući da se program temelji na prepoznavanju više objekata, potrebno je u programu odrediti kada je na sceni prepoznat objekt, a kad ga nema. SURF algoritma dovoljno dobro odradi svoj dio i na sceni pronalazi dovoljan broj ključnih točaka koje se povezuju sa slikom objekta. Povezivanjem objekta i objekta na sceni iscrtavaju se linije koje okružuju objekt na sceni. Ako je objekt zaokružen, možemo reći da ga je program pronašao (slika 26.).



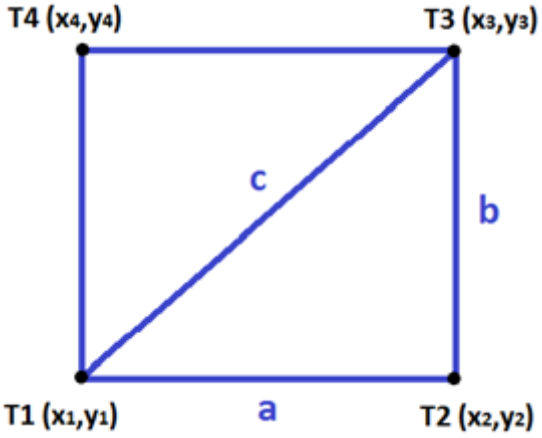
Slika 26. Prepoznata kutija na sceni

Međutim, to je ono što mi vidimo i što možemo zaključiti. Problem je računalu reći da je objekt koji je zaokružen upravo traženi objekt, u ovom slučaju kutija. Taj problem je riješen geometrijskim opisom objekta. Ako pogledamo sliku 21., kutija je opisana četverokutom koji ima svoj geometrijski opis. Upravo na temelju tog četverokuta odredit će se je li na sceni kutija ili nije.

Na slici 27. prikazan je četverokut koji opisuje kutiju. Točke *T1* do *T4* su koordinate u pikselima na slici. Za opisivanje četverokuta uzet će se duljina hipotenuze *c* i duljine stranica *a* i *b*. Pa tako će stranica *a* biti:

$$a = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

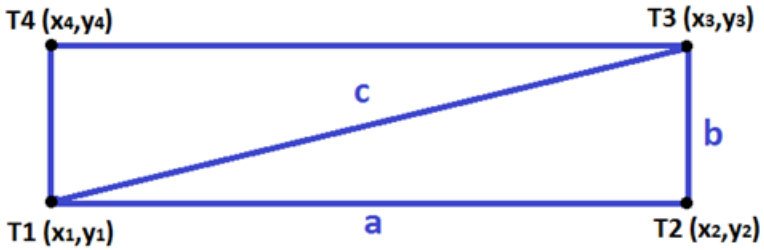
Analogno tomu izračunavaju se duljine stranice *b* i hipotenuze *c*.



Slika 27. Četverokut koji opisuje kutiju

Sada je potrebno postaviti uvjet za koji ako će biti zadovoljen, program ispisuje da je na sceni pronađen objekt. Prvi uvjet je: duljina stranice *c* izračunata Euklidovom formulom preko koordinata i duljina stranice *c* izračunata Pitagorinim pučkom $c = \sqrt{a^2 + b^2}$ moraju biti podjednake s odstupanjima +/- 10%. Drugi uvjet je da stranica *a* mora biti unutar 20% do 30% duljine stranice *b*. Odstupanja su dodana zbog toga jer je homografija temeljena na RANSAC-u koji slučajnim odabirom uzima kandidate iz seta točaka.

Na isti način postavlja se uvjet za opisivanje daljinskoga upravljača. On također ima oblik četverokuta, ali u ovom slučaju dujina stranice *a* je mnogo veća od duljine stranice *b* (slika 28.).



Slika 28. Četverokut koji opisuje daljinski upravljač

Uvjet za duljinu hipotenuze *c* ostaje isti, ali se dodaje uvjet da stranica *a* mora biti 4 puta veća od stranice *b*.

Upravo na temelju tih uvjeta u programu postoji tok grananja. Prvo se za prvu scenu ispituje je li kutija na sceni, potom se ispituje je li daljinski upravljač na sceni. Ako je objekt na sceni, program će prikazati rezultate i ispisati da je pronađen, ako objekta nema, program će ispisati da nije pronađen objekt.

6.8. Analiza SURF algoritma

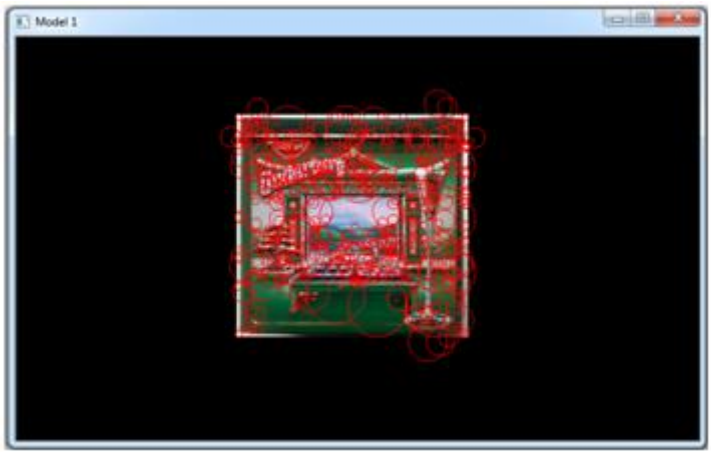
SURF algoritam je pouzdan algoritam za traženje značajnih točaka na slici. Međutim, svaki algoritam ima svoje prednosti i mane, pa tako i SURF. Kod prepoznavanja objekata SURF ima istu prednost i manu, a to je upravo traženje točaka. Prednost je što pronalazi dovoljno velik broj značajnih točaka na temelju kojih se može prepoznati objekt, a u drugu ruku taj prevelik broj pronađenih točaka na sceni može dovesti do problema prepoznavanja, pogotovu kad objekt nije na sceni. Kao što je rečeno, svaka točka je opisana svojim deskriptorom pomoću kojih se vrši povezivanje između slika. Taj nedostatak najviše dolazi na vidjelo ako se objekt nalazi na nehomogenoj, neuređenoj sceni. Ako je objekt na crnoj homogenoj podlozi, SURF će pronaći značajne točke samo na objektu a ne u okolini objekta. Zašto je to tako?

Kako je rečeno, SURF se temelji na pronalaženju blob-struktura s ciljem detektiranja točaka ili regija na slici koja imaju drugačija obilježja u kontrastu svjetloće ili boje u usporedbi s okolinom. To se postiže traženjem lokalnih maksimuma (minimuma) u području intenziteta. Tu je i najveći problem jer su lokalni ekstremi dosta osjetljivi na šumove, što i rezultira

pronalaženjem velikog broja značajki na nehomogenim slikama odnosno u nehomogenoj okolini objekta.

6.8.1. Analiza uređenja okoline objekta

Za početak kutija će biti postavljena na crnu podlogu. Primjenom SURF algoritma pronađene su sljedeće značajke odnosno blob-strukture (slika 29.)



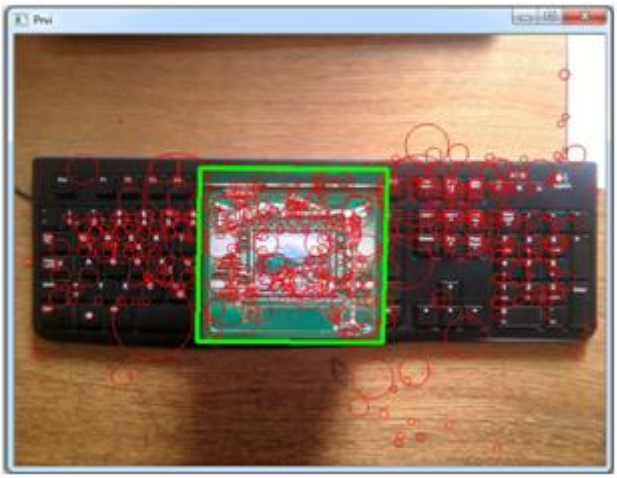
Slika 29. Kutija postavljena na crnu podlogu

Na slici se jasno vidi da su sve značajke pronađene na objektu a nijedna u okolini objekta. Okolina je homogena i nema promjena u kontrastu svjetloće i boje. Jednom riječju, u okolini objekta nema šumova te stoga blob-strukture nisu pronađene.

Na sljedećoj slici vidimo primjenu SURF algoritma kad se objekt nalazi na podlozi koja nije homogena.

Slika 30. Kutija na neuređenoj sceni

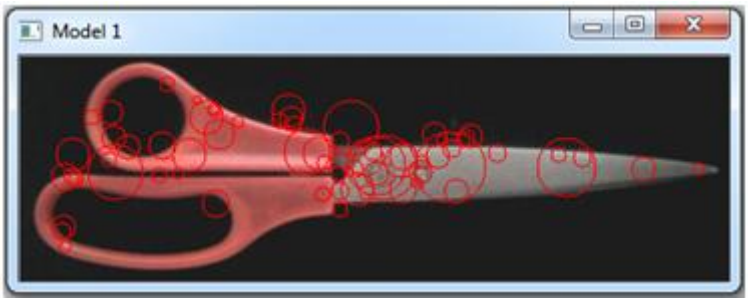
Tu se vidi da je algoritam u okolini kutije pronašao veliki broj dodatnih značajki. Tu može doći do problema prepoznavanja objekta na sceni, tj. kad algoritam nađe preveliki broj značajki u okolini. Međutim, u slučaju kutije na ovoj sceni to ipak nije bio problem, algoritam za povezivanje imao je dovoljan broj deskriptora za povezivanje (slika 31.). A razlog tomu je upravo i količina dobrih značajki pronađenih na kutiji. Na početku sam rekao da sam upravo iz tog razloga odabrao objekte kutiju i daljinski upravljač, jer na sebi imaju izrazito mnogo dobrih značajki, odnosno mnogo područja gdje je veliki kontrast osvjetljenja i boja.



Slika 31. Pronađena kutija na neuređenoj sceni

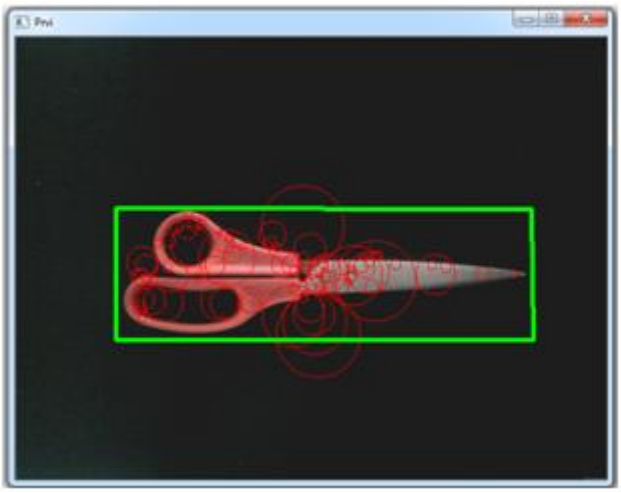
6.8.2. Analiza objekta s premalo značajki

Tu ću se osvrnuti na objekte koji imaju manje značajki. Za primjer će se uzeti škare. Škare po svojoj površini nemaju previše različitosti ali opet neke značajke se mogu pronaći. SURF algoritam je našao manje značajki nego kod primjera s kutijom (slika 32.).



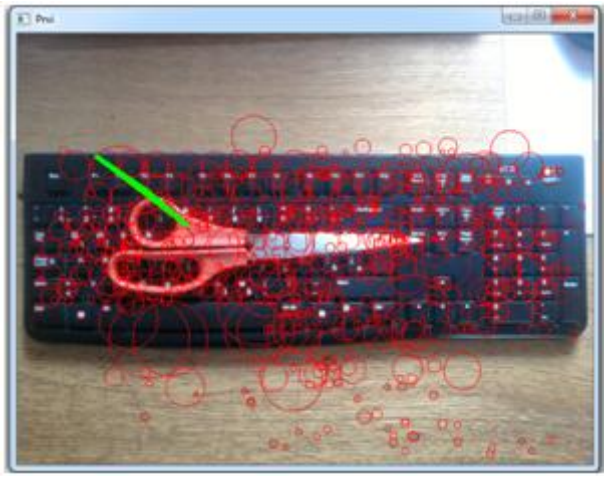
Slika 32. Blob-strukture pronađene na škarama

Sada ću škare staviti na crnu podlogu i pokrenuti program prepoznavanja. Rezultat je prikazana na slici 33.



Slika 33. Pronađene škare na crnoj podlozi

Donekle i očekivani rezultat, program je pronašao škare na crnoj podlozi. No, što će se dogoditi ako škare postavimo na podlogu koja nije homogena. Sljedeća slika prikazuje rezultat toga.

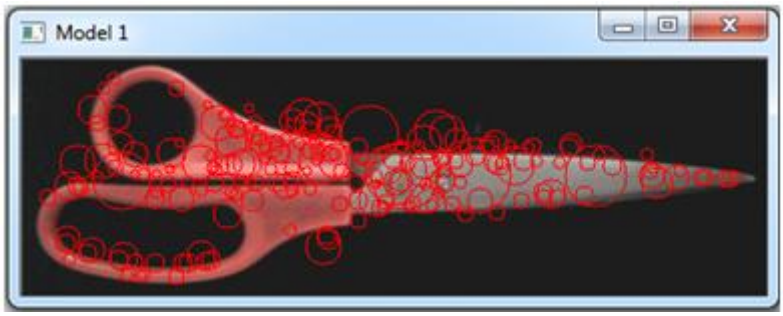


Slika 34. Škare na neuređenoj sceni

Uočava se da algoritam nije uspio pronaći i zaokružiti škare. Ali nešto je ipak pronašao i zaokružio (zelena linija na slici 34.). Scena na kojoj se nalaze škare ima jednostavno previše značajki u okolini škara i algoritam je uspio pronaći neke parove deskriptora, ali ne one potrebne koji bi bili dovoljni za prepoznavanje škara.

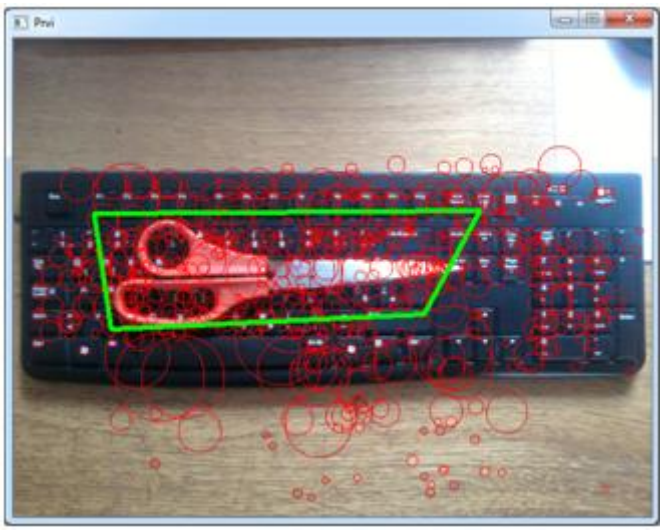
Međutim, jedan od osnovnih parametara SURF algoritma je *hessianTreshold* čija je vrijednost najbolja između 300 i 400. Taj broj predstavlja granicu iznad koje se izdvajaju značajke. Ako je granica manja, na slici se pronalazi više značajki, ako je veća, pronalazi se manje značajki.

U svrhu ispitivanja algoritma za objekt škare granica je postavljena na manju vrijednost (do sada je za sva ispitivanja bila 400, sada je postavljena na 100) i na škarama je pronađeno više značajki, što je prikazano na slici 35.



Slika 35. Pronađene blob-strukture na škarama s *tresholdom* postavljenim na100

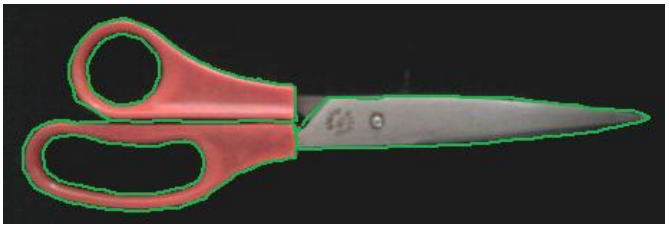
No koliko je dobro podizati i spuštati tu granicu? Ako pokušamo ponovno usporediti objekt škare sa scenom na kojoj se nalaze škare, dobit ćemo sljedeći rezultat prikazan na slici 36.



Slika 36. Donekle pronađene škare na slici

Program je uspio donekle pronaći škare na sceni, ali ipak nedovoljno dobro da bi se za objekt sa sigurnošću moglo reći da se nalazi na sceni. Jer vidimo da nije cijeli objekt zaokružen niti je zaokružen pravokutnikom.

Jedan od pristupa prepoznavanja objekata s manje značajnih točaka bilo bi prepoznavanje objekata preko istaknutih regija. Istaknute regije su elementi slike koje imaju neke istaknute značajke npr. element slike s istom bojom. Najpoznatija metoda za otkrivanje takvih regija, blob-struktura je MSER (*maximally stable extremmal regions*). Te regije su nakupine piksela koje su svjetlije ili tamnije od svih piksela na granici regije. Primjerice, koristeći na škarama MSER, dijelovi s istom bojom bili bi jedna blob-struktura odnosno jedna istaknuta regija (slika 37.). Dakle slično segmentaciji, izvlače se homogene regije koje su postojane preko širokog prostora *tresholda*.

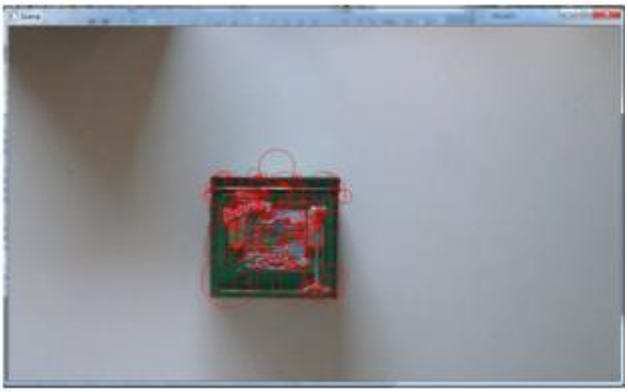


Slika 37. Blob-strukture na škarama pronađene MSER algoritmom

To bi pomoglo u prepoznavanju objekata koji na sebi nemaju previše značajki. Međutim, zadatak diplomskoga bio je prepoznavanje objekata koji na sebi imaju više značajki pa je u svrhu toga korišten SURF algoritam. Osim toga SURF algoritam je mnogo brži od MSER algoritma i jednostavniji je kad je u pitanju povezivanje značajnih točaka.

6.8.3. Analiza zumiranja

Ova se analiza najviše odnosi na kameru. Prilikom udaljavanja gubi se mnogo podataka na slici ako kamera nema dobru rezoluciju. SURF algoritam pronalazi i dalje određeni broj značajki, ali pošto se izgubilo dosta informacija, ne pronalaze se značajke koje se mogu dobro povezati sa značajkama izvorne slike. Primjer udaljavanja kamere od objekta kutije i potom pokretanje programa za prepoznavanje objekta daje rezultat nepronađene kutije na sceni, što je prikazano na slici 38.



Slika 38. Kutija s udaljenim pogledom kamere

6.8.4. Zaključak analize

Program za prepoznavanje najviše ovisi o SURF algoritmu i pronađenim značajkama na sceni. U analizi je viđeno da dvije osnovne stvari utječu na točnost prepoznavanja objekta, to su podloga, tj. pozadina na kojoj se nalazi objekt i granica *hessianTreshold-a*. Smanjenjem granice povećava se broj nađenih značajki. ali to nije od velike pomoći jer algoritam koji povezuje deskriptore ima na raspolaganju mnogo više deskriptora i samim time mogućnost netočnoga povezivanja ako se objekt nalazi na nehomogenoj podlozi. Stoga najveći utjecaj na točnost prepoznavanja objekta ima upravo podloga na kojoj se objekt nalazi. Ako se nalazi na crnoj podlozi, kao što se može vidjeti u rezultatima analize, u okolini objekta se ne pronalaze nikakve značajke, pa algoritam za povezivanje može vrlo točno povezati deskriptore objekta i objekta na crnoj podlozi. Zaključak je: da bi se objekt sa sigurnošću mogao dobro prepoznati, potrebno je urediti podlogu na kojoj se nalazi i odabrati granicu *hessianTreshold-a* za analiziranje scene upravo kako je i predloženo u algoritmu između 300 i 500. Za traženje blob-struktura na izvornoj slici objekta koji se želi pronaći i prepoznati, granica *hessianTreshold-a* se može i spustiti ispod 300, pogotovu ako objekt na sebi nema dovoljno značajki. Vrlo je važno pri tome imati i dobru kameru s visokom rezolucijom.

7. SPAJANJE S IP KAMEROM

Da bi se upotpunio zadatak, potrebno je programsko rješenje povezati s dostupnom IP kamerom.

7.1. VLC

Za početak je najprije potrebno odrediti na koji način će se uzimati slike s kamere i spremati u željenu datoteku. OpenCV podržava *FFmpeg* (*record, convert and stream audio and video*) biblioteku, međutim to još nije dobro riješeno u dosadašnjim verzijama OpenCV-a pa se tražilo drugo rješenje spremanja slika s kamere.

Rješenje je pronađeno u jednostavnom programu *VLC* (*video lan*). *VLC* je besplatni program, otvorenoga koda za reprodukciju medija i reprodukciju preko interneta. Podržava mnoge video i audio-formate te protokole za prijenos podataka preko interneta. Sadržava veliki broj biblioteka za kodiranje i dekodiranje. Mnogi koderi/dekoderi su iz libavcodec biblioteke koja je dio *FFmpeg-a.*

Kod za uzimanje slika s kamere nalazit će se u *php* skripti, koja će pokretati i program za prepoznavanje objekata.

IP kamera se nalazi na serveru pa će se preko servera i pristupati kameri. Za to je potrebna instalacija softvera koji će nam to omogućiti.

7.2. WAMP server

WAMP server je windows razvojno okruženje koje omogućuje kreiranje mrežnih aplikacija sa bazama podataka kao što su *Apache2, PHP, MySQ*L. Na *WAMP* server možemo pristupati preko bilo kojega preglednika i s bilo kojeg udaljenog računala. To daje veliku prednost jer nismo vezani za određenu lokaciju. Na *WAMP* serveru će se nalaziti direktoriji u koji će biti pohranjena *php* skripta i aplikacija *PTZ_control_polgar .bat* koja će pokretati *php* skriptu.

7.3. PHP

Unutar *php* skripte nalazi se konačno rješenje zadatka. Grafički prikaz koda unutar skripte prikazan je na sljedećoj slici.

Slika 39. Dijagram toka izvođenja zadatka diplomskoga rada

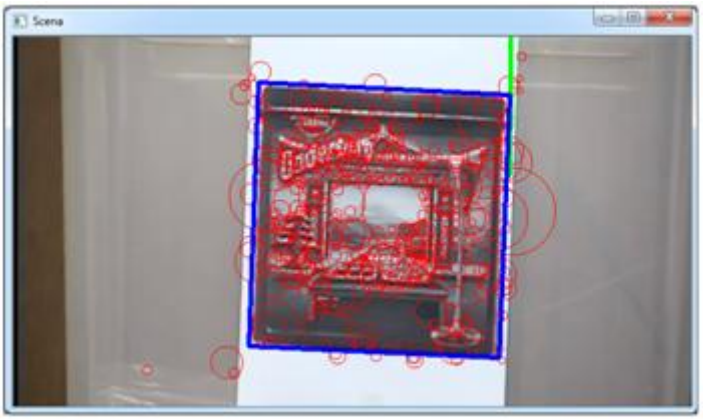
7.3.1. Opis izvođenja php skripte

Pokretanjem skripte kamera dolazi u poziciju 1. Pokreće se VLC koji učitava kameru i uzima nekoliko kadrova pozicije 1. Slike se spremaju u željeni direktorij i izlazi se iz VLC-a. Nakon toga pokreće se program za prepoznavanje objekata koji učitava prvu sliku scene s prve pozicije kamere te ju obrađuje i izvršava zadatak prepoznavanja te prikazuje rezultat na zaslonu računala. Po završetku programa za prepoznavanje petlja u *php* skripti se povećava za 1, odnosno kamera dolazi u poziciju 2 i ponovo slijedi opisani dio. Kad kamera prođe sve četiri pozicije prekida se php skripta i završava izvođenje programa.

8. REZULTATI PRETRAGE SCENA

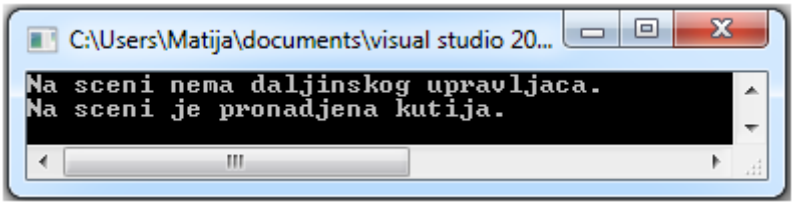
Primjenom programa za prepoznavanje objekata na scenama dobivaju se sljedeći rezultati:

Za scenu 1 na kojoj se nalazi kutija SURF algoritam pronalazi ključne točke koje se povezuju s ključnim točkama izvorne slike kutije. Rezultat je prikazan na slici 40.



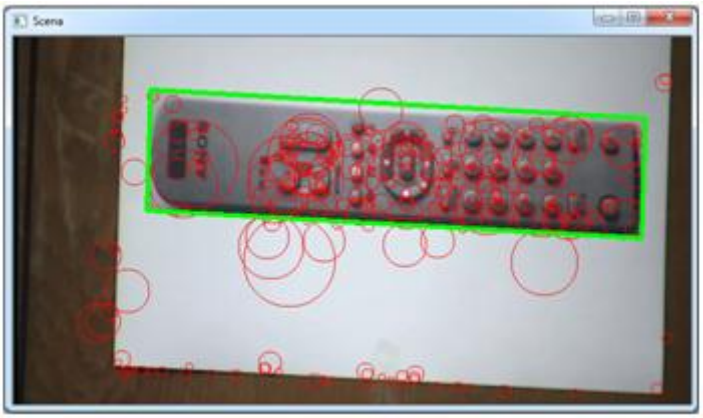
Slika 40. Pronađena kutija na sceni 1.

Algoritam je našao dovoljno ključnih točaka za povezivanje i sa 4 linije zaokružen je pronađeni objekt. Uz to, program na zaslonu ispisuje nalazi li se objekt ili ne nalazi na sceni, slika 41.



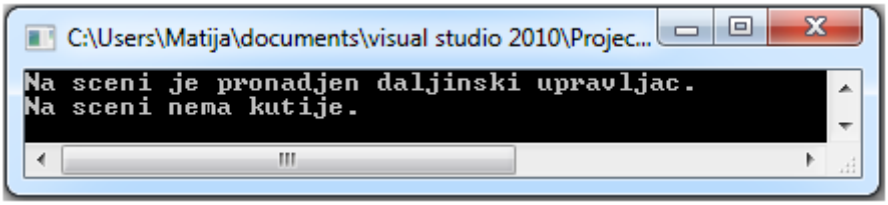
Slika 41. Ispis rezultata pretrage scene 1

Sljedeća scena je scena 2, s daljinskim upravljačem. Isto kao i za prethodnu scenu, SURF algoritam pronalazi ključne točke i povezuje ih s ključnim točkama izvorne slike daljinskog upravljača. Rezultat je prikazan na slici 42. uz ispis na zaslonu, slika 43.



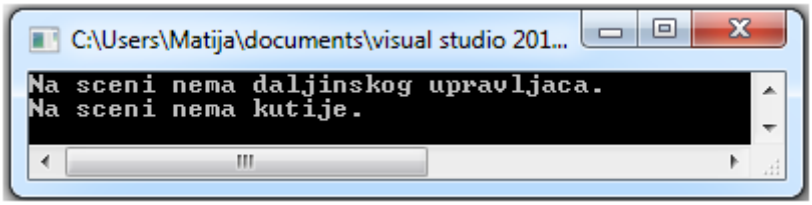
Slika 42. Pronađen daljinski upravljač na sceni 2

Na sceni 2 možemo primijetiti kako je SURF algoritam pronašao više blob-struktura u okolini objekta, međutim, na daljinskom upravljaču je pronašao dovoljno dobrih ključnih da bi se mogao povezati s izvornom slikom, što je i prikazano zaokruživanjem objekta sa 4 zelene linije.



Slika 43. Ispis rezultata pretrage scene 2

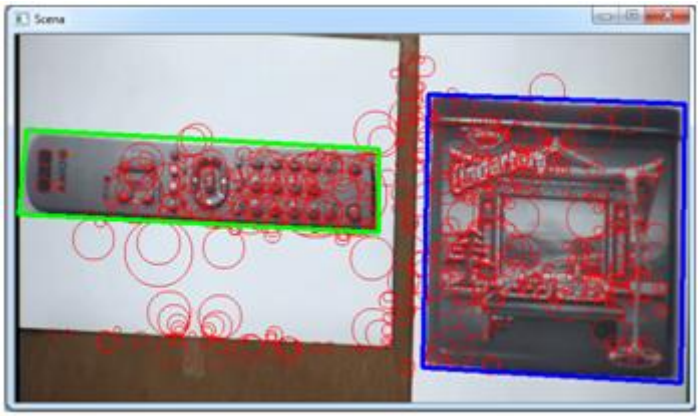
Scena 3 je scena bez ijednog objekta na kojoj nisu ni pronađene blob-strukture, stoga ni objekt. Ispis rezultata pretrage na sceni tri je prikazn na slici 44.



Slika 44. Ispis rezultata pretrage scene 3

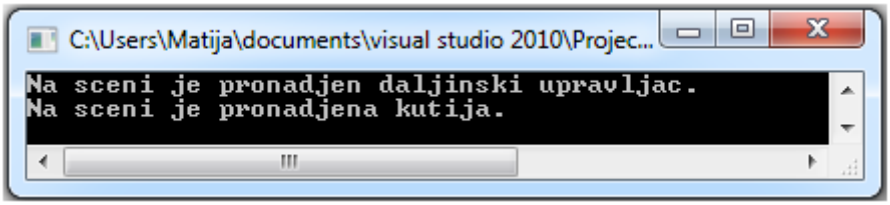
Zadnja scena koja se pretražuje je scena 4, s oba objekta. Na slici 45. vidimo rezultat pretrage te scene. Pronađena su i zaokružena linijama oba objekta. Isto kao i na sceni 2, SURF

algoritam je pronašao i veći broj blob-struktura u okolini objekata, ali opet je našao dovoljno ključnih objekata na točkama za uspješno povezivanje i pronalaženje objekta. To sam već napomenuo da je razlog upravo taj što kutija i daljinski upravljač imaju puno značajki po svojoj površini.



Slika 45. Pronađena kutija i daljinski upravljač na sceni 4.

Slijedi ispis rezultata, što je prikazano na slici 46.



Slika 46. Ispis rezultata pretrage scene 4

Kao što možemo vidjeti, na prikazanim rezultatima program je pronašao svaki put objekt kad se nalazi na sceni. Uz prikazane rezultate i prijašnju analizu SURF algoritma mogu reći da je program pouzdan i da će svaki put pronaći objekt kad se nalazi na uređenoj podlozi. Pa čak i kad podloga nije uređena, ako objekt na sebi ima puno značajki, velika je vjerojatnost da bude pronađen.

9. ZAKLJUČAK

Diplomski se rad sastojao od izradbe programskoga rješenja za prepoznavanje više objekata i povezivanje istoga s IP kamerom. Pri tome su primjenjivane metode i algoritmi koji se godinama razvijaju za tu svrhu.

U radu se pokazalo kako prepoznavanje objekata nije jednostavan problem. Iako računalo posjeduje umjetnu inteligenciju, nema svijest niti inteligenciju kao čovjek. Niz matematičkih operacija koje se obrađuju u procesoru nisu dovoljne za rješavanje složenijih stvari koje čovjek obavlja instinktivno i s lakoćom.

Konkretno, ovaj rad se bavio problemima računalnoga vida. U svrhu toga koristile su se kombinacije algoritama pomoću kojih se tražilo rješenje problema prepoznavanja objekata. Prije svega može se reći da je najvažniji algoritam SURF. Njegova funkcija je pronalaženje ključnih točaka na slikama i konstruiranje deskriptora. U radu je prikazana analiza tog algoritma. Iako je jedan od boljih algoritama, pogotovu kad su posrijedi brzina i kvaliteta pronalaženja točaka, kao i svaki algoritam ima svoje mane. Tu je najvažnije naglasiti problem objekata s premalo značajki i osjetljivost na šumove, budući da je pronalaženje temeljeno na promjenama u kontrastu slike, bilo osvjetljenja ili boje. Stoga, kako je i u analizi prikazano, na nehomogenim scenama dolazi do određenih poteškoća kad je u pitanju prepoznavanje objekata. No, ipak, u ovom diplomskom radu, objekti koje je trebalo prepoznati na sebi su imali puno značajki, što je za ovaj problem uporaba SURF algoritma bila dovoljno dobra. Pronalaženjem ključnih točaka, njihovih parova pomoću k-NN algoritma i povezivanjem točaka homografijom u radu je riješen problem prepoznavanja objekata.

Povezivanje programa s kamerom nije stvaralo previše poteškoća. Budući da se kameri nije moglo pristupati izravno preko programa za prepoznavanje objekata, korišten je jednostavni i besplatni program VLC koji ima mogućnost prijenosa zapisa preko *RTSP* protokola.

Svi programi i biblioteke funkcija s algoritmima koje su korištene dostupne su bez naplaćivanja i mogu reći da su dovoljno dobre za rješavanje mnogih problema računalnoga vida, ali ipak ne još uvijek toliko sofisticirane. Međutim, na tom području se i dalje intenzivno radi i u skoroj budućnosti možemo očekivati daleko bolja i efikasnija rješenja kad se radi o problemima računalnoga vida pa i ostalim problemima na području umjetne inteligencije.

PRILOZI

I. Kod programa za prepoznavanje objekata
II. Kod PHP skripte
III. CD-R disk

LITERATURA

[1] Szeliski, R.: Computer Vision: Algorithms and Applications, Springer, London, 2011.

[2] Bradski, G., Kaeher, A.: Learning OpenCV, O+Reilly Media, Inc., Debastopol, 2008.

[3] Dubrofsky, E.: Hohography Estimation, The University of British Coulumbia, Vancouver, 2009.

[4] Bay, H., Ess, A., Tuytelaars, T., Gool, L.: Speeded-Up Robust Features (SURF), ETH Zurich, Zurich, 2008.

[5] Bently, J. L.: Multidimensional Binary Search Trees Used for Associative Searching, Stanford University, Stanford, 1957.

[6] Garcia, V., Debreuve, E., Nielsen, F., Barlaud, M.: K-Nearest Neighbor Search: Fast GPU-Based Implementations and Application to High-Dimensional Feature Matching, IEEE, ICP, 2010.

[7] Fomby, T. B.: K-Nearest Neighobrs Search: Prediction and Classification, Southern Metodist University, Dallas, 2008.

[8] OpenCV: http://opencv.willowgarage.com/wiki/, 2102.

[9] Wikipedia: http://en.wikipedia.org/wiki/K-d_tree, 2012.

[10] Wikipedia: http://en.wikipedia.org/wiki/IP_camera, 2012.

I. KOD PROGRAMA ZA PREPOZNAVANJE VIŠE OBJEKATA

```
//učitavanje biblioteka
#include <stdio.h>
#include <iostream>
#include <cxcore.h>
#include <cv.h>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include <cstdlib>
#include <math.h>
#include <ctype.h>
#include <vector>


using namespace cv;
using namespace std;
#define USE_FLANN

//deklariranje globalnih varijabli
IplImage* scena = 0;
IplImage* scena_GREY = 0;
IplImage* model_2 = 0;
IplImage* model_2_GREY = 0;
IplImage* model_1 = 0;
IplImage* model_1_GREY = 0;

double a;
double b;
double c;
double c1;

int p1x[3];
int p1y[3];
int p2x[3];
int p2y[3];

// deklariranje funkcija
void flannFindPairs( const CvSeq*, const CvSeq* objectDescriptors, const CvSeq*, const
CvSeq* imageDescriptors, vector<int>& ptpairs );
int locatePlanarObject( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
const CvSeq* imageKeypoints, const CvSeq* imageDescriptors, const CvPoint
src_corners[4], CvPoint dst_corners[4] );



//glavna funkcija
int main(){
	//kreiranje prozora
	cvNamedWindow("Scena");
	cvNamedWindow("Model1");
	cvNamedWindow("Model2");

	//deklariranje parametara cvExtraxtSurf i spremišta
	CvMemStorage* spremiste = cvCreateMemStorage(0);
	CvSURFParams param1 = cvSURFParams(400, 1);
	//deklariranje ključnih točaka i deskriptora
	CvSeq *tocke = 0, *deskriptori = 0;
	CvSeq *tocke_2 = 0, *deskriptori_2 = 0;
	CvSeq *tocke_1 = 0, *deskriptori_1 = 0;
```

```
	//učitavanje slika
	model_2 = cvLoadImage("C:\\Model\\model_2.jpg");
	model_1 = cvLoadImage("C:\\Model\\model_1.jpg");
	scena = cvLoadImage("C:\\Grab\\img-00001.png");

	//pretvaranje slika u 8-bitne slike
	model_2_GREY = cvCreateImage(cvGetSize(model_2),8,1);
	model_1_GREY = cvCreateImage(cvGetSize(model_1),8,1);
	scena_GREY = cvCreateImage(cvGetSize(scena),8,1);

	//pretvaranje slika u crno-bijele slike
	cvCvtColor(scena,scena_GREY,CV_RGB2GRAY);
	cvCvtColor(model_2,model_2_GREY,CV_RGB2GRAY);
	cvCvtColor(model_1,model_1_GREY,CV_RGB2GRAY);

	// traženje ključnih točaka i računanje deskriptora
	cvExtractSURF(model_2_GREY, 0, &tocke_2, &deskriptori_2, spremiste, param1, 0);
	cvExtractSURF(model_1_GREY, 0, &tocke_1, &deskriptori_1, spremiste, param1, 0);
	cvExtractSURF(scena_GREY, 0, &tocke, &deskriptori, spremiste, param1, 0);

	//određivanje kutnih točaka modela 1 i modela 2
	CvPoint src_corners[4] = {{0,0}, {model_2_GREY->width,0}, {model_2_GREY->width,
model_2_GREY->height}, {0, model_2_GREY->height}};
	CvPoint src_corners1[4] = {{0,0}, {model_1_GREY->width,0}, {model_1_GREY->width,
model_1_GREY->height}, {0, model_1_GREY->height}};
	CvPoint dst_corners[4];
	CvPoint dst_corners1[4];

	//zaokruživanje ključnih točaka na slikama
		for (int i = 0; i < tocke->total; i++){
					CvSURFPoint* r =
(CvSURFPoint*)cvGetSeqElem(tocke, i);
					CvPoint centar;
					int radius;
					centar.x = cvRound(r->pt.x);
					centar.y = cvRound(r->pt.y);
					radius = cvRound(r ->size*1.2/9.*2);
					cvCircle(scena, centar, radius,
cvScalar(0,0,255),1, 8, 0);
					}

		for (int i = 0; i < tocke_2->total; i++){
					CvSURFPoint* r =
(CvSURFPoint*)cvGetSeqElem(tocke_2, i);
					CvPoint centar;
					int radius;
					centar.x = cvRound(r->pt.x);
					centar.y = cvRound(r->pt.y);
					radius = cvRound(r ->size*1.2/9.*2);
					cvCircle(model_2, centar, radius,
cvScalar(0,0,255),1, 8, 0);
					}

		for (int i = 0; i < tocke_1->total; i++){
					CvSURFPoint* r =
(CvSURFPoint*)cvGetSeqElem(tocke_1, i);
					CvPoint centar;
					int radius;
					centar.x = cvRound(r->pt.x);
					centar.y = cvRound(r->pt.y);
					radius = cvRound(r ->size*1.2/9.*2);
```

```
                                        cvCircle(model_1, centar, radius,
cvScalar(0,0,255),1, 8, 0);
                                        }

              //unutar ove funkcije pretražuju se i povezuju parovi
               if( locatePlanarObject( tocke, deskriptori, tocke_2,
        deskriptori_2, src_corners, dst_corners ))
    {
        for(int i = 0; i < 4; i++ )
        {
                    //iscrtavanje 4 linije oko pronađenog objekta na sceni
            CvPoint r1 = dst_corners[i%4];
            CvPoint r2 = dst_corners[(i+1)%4];
            cvLine( scena, cvPoint(r1.x, r1.y),
            cvPoint(r2.x, r2.y), cvScalar(0,255,0),3 );
                    //unošenje podataka u polja
                    p1x[i]=r1.x;
                    p1y[i]=r1.y;
                    p2x[i]=r2.x;
                    p2y[i]=r2.y;

             }
    }
             //računanje udaljenosti između točaka
             c = (p1x[0]-p2x[1])*(p1x[0]-p2x[1]) + (p1y[0]-p2y[1])*(p1y[0]-p2y[1]);
             c = sqrt(c);
             a = (p1x[0]-p2x[0])*(p1x[0]-p2x[0]) + (p1y[0]-p2y[0])*(p1y[0]-p2y[0]);
             a = sqrt(a);
             b = (p2x[0]-p2x[1])*(p2x[0]-p2x[1]) + (p2y[0]-p2y[1])*(p2y[0]-p2y[1]);
             b = sqrt(b);
             c1 = a*a + b*b;
             c1 = sqrt(c1);

             /*cout << c << endl;
             cout << c1 << endl;
             cout << a << endl;
             cout << b << endl;*/

//postavljanje uvjeta na temelju kojeg se određuje da li je objekt pronađen
             if (0.9*c1 < c && c < 1.1*c1 && a > 4*b && a < 5*b){
                    cout << "Na sceni je pronadjen daljinski upravljac." << endl;

                    cvShowImage("Scena", scena);
                    cvShowImage("Model2", model_2);
             }
             else{
                    cout << "Na sceni nema daljinskog upravljaca." << endl;

                    cvShowImage("Scena", scena);
             }
             if ( locatePlanarObject( tocke, deskriptori, tocke_1,
        deskriptori_1, src_corners1, dst_corners1 ))
    {
        for(int i = 0; i < 4; i++ )
        {

            CvPoint r1 = dst_corners1[i%4];
            CvPoint r2 = dst_corners1[(i+1)%4];
            cvLine( scena, cvPoint(r1.x, r1.y),
            cvPoint(r2.x, r2.y), cvScalar(255,0,0),3 );
```

```
                        p1x[i]=r1.x;
                        p1y[i]=r1.y;
                        p2x[i]=r2.x;
                        p2y[i]=r2.y;
                }
    }

                c = (p1x[0]-p2x[1])*(p1x[0]-p2x[1]) + (p1y[0]-p2y[1])*(p1y[0]-p2y[1]);
                c = sqrt(c);
                a = (p1x[0]-p2x[0])*(p1x[0]-p2x[0]) + (p1y[0]-p2y[0])*(p1y[0]-p2y[0]);
                a = sqrt(a);
                b = (p2x[0]-p2x[1])*(p2x[0]-p2x[1]) + (p2y[0]-p2y[1])*(p2y[0]-p2y[1]);
                b = sqrt(b);
                c1 = a*a + b*b;
                c1 = sqrt(c1);


                /*cout << "a = " <<a << endl;
                cout << "b = " <<b << endl;
                cout << "c = " <<c << endl;
                cout << "c1 = " <<c1 << endl;*/ //računanje duljina


                if (0.9*c1 < c && c < 1.1*c1 && 0.9*b < a && a< 1.1*b){


                cout << "Na sceni je pronadjena kutija." << endl;

                cvShowImage("Scena", scena);
                cvShowImage("Model1", model_1);

                }
                else{
                        cout << "Na sceni nema kutije." << endl;

                }
waitKey(0);



return 0;
}



// funkcija koja traži parove na temelju strukturiranja k-d stablima i pretragom uz
pomoć k-nn algoritma
void flannFindPairs( const CvSeq*, const CvSeq* objectDescriptors,
            const CvSeq*, const CvSeq* imageDescriptors, vector<int>& ptpairs )
{
        int length = (int)(objectDescriptors->elem_size/sizeof(float));

    cv::Mat m_object(objectDescriptors->total, length, CV_32F);
       cv::Mat m_image(imageDescriptors->total, length, CV_32F);



    CvSeqReader obj_reader;
       float* obj_ptr = m_object.ptr<float>(0);
cvStartReadSeq( objectDescriptors, &obj_reader );
    for(int i = 0; i < objectDescriptors->total; i++ )
    {
```

```
        const float* descriptor = (const float*)obj_reader.ptr;
        CV_NEXT_SEQ_ELEM( obj_reader.seq->elem_size, obj_reader );
        memcpy(obj_ptr, descriptor, length*sizeof(float));
        obj_ptr += length;
    }
    CvSeqReader img_reader;
       float* img_ptr = m_image.ptr<float>(0);
    cvStartReadSeq( imageDescriptors, &img_reader );
    for(int i = 0; i < imageDescriptors->total; i++ )
    {
        const float* descriptor = (const float*)img_reader.ptr;
        CV_NEXT_SEQ_ELEM( img_reader.seq->elem_size, img_reader );
        memcpy(img_ptr, descriptor, length*sizeof(float));
        img_ptr += length;
    }


    cv::Mat m_indices(objectDescriptors->total, 2, CV_32S);
    cv::Mat m_dists(objectDescriptors->total, 2, CV_32F);
    cv::flann::Index flann_index(m_image, cv::flann::KDTreeIndexParams(4));  //
koristi 4 nasumična k-d stabla
    flann_index.knnSearch(m_object, m_indices, m_dists, 2, cv::flann::SearchParams(64)
); // maksimalni provjereni broj listova

    int* indices_ptr = m_indices.ptr<int>(0);
    float* dists_ptr = m_dists.ptr<float>(0);
    for (int i=0;i<m_indices.rows;++i) {
       if (dists_ptr[2*i]<0.6*dists_ptr[2*i+1]) {
              ptpairs.push_back(i);
              ptpairs.push_back(indices_ptr[2*i]);

       }

    }



}



//funkcija pretražuju se i povezuju parovi ako je definiran FLANN
int
locatePlanarObject( const CvSeq* objectKeypoints, const CvSeq* objectDescriptors,
                    const CvSeq* imageKeypoints, const CvSeq* imageDescriptors,
                    const CvPoint src_corners[4], CvPoint dst_corners[4] )
{
    double h[9];
       double A[9];
    CvMat _h = cvMat(3, 3, CV_64F, h);
       CvMat _A = cvMat(3, 3, CV_64F, A);
    vector<int> ptpairs;
    vector<CvPoint2D32f> pt1, pt2;
    CvMat _pt1, _pt2;
    int i, n;

#ifdef USE_FLANN
    flannFindPairs( objectKeypoints, objectDescriptors, imageKeypoints,
imageDescriptors, ptpairs );

#endif
```

```
    n = ptpairs.size()/2;
    if( n < 4 )
        return 0;

    pt1.resize(n);
    pt2.resize(n);
    for( i = 0; i < n; i++ )
    {
        pt1[i] = ((CvSURFPoint*)cvGetSeqElem(objectKeypoints,ptpairs[i*2]))->pt;
        pt2[i] = ((CvSURFPoint*)cvGetSeqElem(imageKeypoints,ptpairs[i*2+1]))->pt;
    }

    _pt1 = cvMat(1, n, CV_32FC2, &pt1[0] );
    _pt2 = cvMat(1, n, CV_32FC2, &pt2[0] );
   // traženje homografije
       if( !cvFindHomography( &_pt1, &_pt2, &_h, CV_RANSAC, 5 ))
        return 0;


      cvInvert(&_h, &_A, CV_LU);
    for( i = 0; i < 4; i++ )
    {
        double x = src_corners[i].x, y = src_corners[i].y;
        double Z = 1./(A[6]*x + A[7]*y + A[8]);
        double X = (A[0]*x + A[1]*y + A[2])*Z;
        double Y = (A[3]*x + A[4]*y + A[5])*Z;
        dst_corners[i] = cvPoint(cvRound(X), cvRound(Y));

    }


    return 1;
}
```

II. KOD PHP SKRIPTE

```
<?php

include('WSSecurity.class.php');
$url = "http://192.168.0.124/onvif/services";

$client = new WSSoapClient(null, array(
'location' => 'http://192.168.0.124/onvif/services',
'uri'      => 'http://www.onvif.org/ver20/ptz/wsdl',
//'soap_version' => 0,
 'trace'      => 1
));

 $client->__setUsernameToken('admin','admin');
 $params = array();
 for ($i=10;$i<14;$i++){
$result = $client->__soapCall
(
    'GotoPreset', array
                  (
                    new SoapParam
                    (
                        new SoapVar(
'<ns1:ProfileToken>profile_cam1_stream1</ns1:ProfileToken>',XSD_ANYXML),'ProfileToken'
                    )
                    ,new SoapParam
                    (
                        new SoapVar('
<ns1:PresetToken>preset'.$i.'</ns1:PresetToken>',XSD_ANYXML),'PresetToken'
                    )
                )
);
sleep (2);

system ('"C:\Program Files (x86)\VideoLAN\VLC\vlc.exe"
"rtsp://192.168.0.124:554/snl/live/1/1/Ux/sido=-Ux/sido=" --video-filter=scene --
vout=dummy -I dummy --dummy-quiet --run-time=3 --scene-ratio=1 --scene-prefix=img- --
scene-path=C:\Grab\ vlc://quit');
sleep (2);
system('"C:\\Obj_rec\\Obj_rec.exe"');
sleep (1);

 }

?>
```